



DMS-170

**FORTTRAN DATA BASE FACILITY
VERSION 1
APPLICATION PROGRAMMING
USER'S GUIDE**

**CDC® OPERATING SYSTEMS:
NOS 1
NOS/BE 1**

REVISION RECORD

Revision

Description

A (03/06/81)

Original release at PSR Level 528.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

© COPYRIGHT CONTROL DATA CORPORATION 1981
All Rights Reserved
Printed in the United States of America

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

<u>Page</u>	<u>Revision</u>
Front Cover	-
Inside Front Cover	A
Title Page	-
ii	A
iii/iv	A
v	A
vi	A
vii	A
viii	A
ix	A
1-1 thru 1-6	A
2-1 thru 2-5	A
3-1 thru 3-15	A
4-1 thru 4-7	A
5-1 thru 5-22	A
6-1	A
6-2	A
A-1 thru A-4	A
B-1 thru B-3	A
C-1 thru C-14	A
Index-1 thru -3	A
Comment Sheet	A
Mailer	-
Back Cover	-



PREFACE

The DMS-170 data management system clearly defines two roles: the role of a data administrator who develops, controls, and maintains the physical data base; and the role of an application programmer who accesses and manipulates the data within that data base. Although the two roles differ considerably, each role requires a knowledge of the tasks being performed by the other. The data administrator, for example, cannot develop a data base without first understanding what type of applications will be required. The application programmer, on the other hand, cannot successfully access data without first understanding how the data is described and what specific controls have been established.

This guide describes the role of the FORTRAN 5 application programmer who is accessing data within a DMS-170 controlled data base environment. The presence of a data administrator is assumed, and the functions associated with that position are described as they directly affect the application programmer.

You should note that appendix C, entitled The Sample Application, is particularly important. This appendix sets up a working environment complete with stored data for use with sample programs. This environment can be duplicated to provide a better understanding of DMS-170 and the tools that are used to create a total data management system.

As described in this publication, DMS-170 operates under control of the following operating systems:

- NOS 1 for the CONTROL DATA CYBER 170 Series; CYBER 70 Models 71, 72, 73, and 74; and 6000 Series Computer Systems.
- NOS/BE 1 for the CDC CYBER 170 Series; CYBER 70 Models 71, 72, 73, and 74; and 6000 Series Computer Systems.

The NOS Manual Abstracts and the NOS/BE Manual Abstracts are instant-sized manuals containing brief descriptions of the contents and intended audience of all NOS and NOS product set manuals, and NOS/BE and NOS/BE product set manuals, respectively. The abstracts manuals can be useful in determining which manuals are of greatest interest to you. The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

As a FORTRAN 5 application programmer, you can find additional pertinent information in the listed Control Data Corporation publications. These publications are listed alphabetically in groupings that indicate relative importance to you as readers of this guide.

The following manuals are of primary interest:

<u>Publication</u>	<u>Publication Number</u>
FORTRAN Data Base Facility Version 1 Reference Manual	60482200
FORTRAN Version 5 Reference Manual	60481300

The following manuals are of secondary interest:

<u>Publication</u>	<u>Publication Number</u>
CYBER Database Control System Version 2 Reference Manual	60481800
NOS Version 1 Manual Abstracts	84000420
NOS Version 1 Reference Manual, Volume 1 of 2	60435400
NOS/BE Version 1 Manual Abstracts	84000470
NOS/BE Version 1 Reference Manual	60493800
Software Publications Release History	60481000

CDC manuals can be ordered from Control Data Corporation,
Literature and Distribution Services, 308 North Dale Street,
St. Paul, Minnesota 55103.

This manual describes a subset of the features
and parameters documented in the FORTRAN Data
Base Facility Version 1 Reference Manual.
Control Data cannot be responsible for the
proper functioning of any features or
parameters not documented in the FORTRAN Data
Base Facility Version 1 Reference Manual.

CONTENTS

NOTATIONS	ix	4. ERROR PROCESSING AND STATUS HANDLING TECHNIQUES	4-1
1. FORTRAN PROGRAMMING WITHIN DMS-170	1-1	Using ERR and END Processing Options	4-1
System Components	1-1	Establishing a Data Base Status Block	4-2
Data Description Language	1-1	Error Checking	4-2
The Schema	1-1	Status Checking	4-2
The Sub-Schema	1-1	Defining Recovery Points	4-4
FORTRAN Data Manipulation Language	1-2	Avoiding Constraint Violations	4-4
CYBER Database Control System	1-2	Anticipating Deadlock Situations	4-6
Master Directory	1-2		
CDCS Batch Test Facility	1-2		
Data Base Procedures	1-3	5. DEVELOPING FORTRAN PROGRAMS	5-1
CYBER Record Manager	1-3	Developing an Application Program	5-1
File Organization	1-3	Compiling and Executing the Source Program	5-1
Multiple-Index Processing	1-3	Sample Programs	5-3
Special Features	1-4		
Concurrency	1-4	6. USING THE CDCS BATCH TEST FACILITY	6-1
File Privacy	1-4	Requirements	6-1
Relations	1-4	Obtaining Load Maps	6-1
Constraints	1-4	Executing the CDCS Batch Test Facility	6-1
Recovery	1-4		
Summary of DMS-170 Components and Features	1-4		
2. ACCESSING THE DATA BASE	2-1	APPENDIXES	
Interpreting the FORTRAN Sub-Schema	2-1	A Standard Character Sets	
FORTRAN Data Manipulation Language	2-1	B Glossary	
DML Language Components	2-1	C The Sample Application	
Syntax Requirements	2-1		
Statement Positioning	2-1	INDEX	
3. PROCESSING THE DATA	3-1	FIGURES	
Using DML to Access the Data Base	3-1	1-1 Schema and Sub-Schema Generation	1-2
Identifying the Sub-Schema	3-1	1-2 CYBER Record Manager Interface	1-3
Establishing the Interface With CDCS	3-2	2-1 A Basic FORTRAN Sub-Schema	2-2
Satisfying Privacy Requirements	3-3	2-2 A Relational FORTRAN Sub-Schema	2-3
Opening a Realm	3-3	2-3 DML Statement Positioning	2-5
Locking/Unlocking a Realm	3-3	3-1 Sub-Schema AVERAGE	3-2
Closing a Realm	3-4	3-2 Identifying the Sub-Schema	3-2
Terminating the Interface With CDCS	3-4	3-3 Establishing the Interface With CDCS	3-3
Using DML to Manipulate Data	3-5	3-4 Satisfying Privacy Requirements	3-3
Writing a Record	3-5	3-5 Opening a Realm	3-4
Reading a Record	3-6	3-6 Locking/Unlocking a Realm	3-4
Sequential Read	3-6	3-7 Closing a Realm	3-5
Random Read	3-6	3-8 Terminating the Interface With CDCS	3-5
Positioning a Realm	3-6	3-9 Writing a Record	3-5
Rewriting a Record	3-7	3-10 Reading Sequentially	3-6
Deleting a Record	3-8	3-11 Reading Randomly	3-6
Using DML to Process Relations	3-8	3-12 Positioning a Realm	3-7
Structure of a Relation	3-9	3-13 Rewriting a Record	3-8
Using the Sub-Schema	3-10	3-14 Deleting a Record	3-9
Opening a Relation	3-10	3-15 Tree Structure and Ranks of a	
Closing a Relation	3-10	Three-Realm Relation	3-9
Reading a Relation	3-12	3-16 Sub-Schema COMPARE	3-11
Sequential Relation Read	3-12	3-17 Tree Structure of Record Occurrences	3-13
Random Relation Read	3-13	3-18 Reading a Relation Sequentially	3-13
Control Break	3-13	3-19 Reading a Relation Randomly	3-13
Null Occurrence	3-13	3-20 Null Record Occurrence Examples	3-14
Positioning a Relation	3-14	3-21 Positioning a Relation	3-15
Updating Realms Joined in a Relation	3-15		

4-1	Establishing a Data Base Status Block	4-2	5-9	Program ADMIT	5-18
4-2	Defining Recovery Points	4-4	6-1	CDCSBTF Control Statement Format	6-1
4-3	Single-File Constraint Example	4-5	6-2	Sample FORTRAN Execution of CDCS	
4-4	Two-File Constraint Example	4-5		Batch Test Facility	6-2
4-5	Deadlock Processing	4-7			
5-1	FORTTRAN DML Preprocessing	5-1			
5-2	DML Control Statement	5-2			
5-3	Executing DML and Compiling the				
	Source Program	5-3			
5-4	Compiling and Executing the				
	Source Program	5-3			
5-5	Program RATING	5-4			
5-6	Program INDAVGE	5-7			
5-7	Program RELATE	5-11			
5-8	Program CHARGES	5-15			

TABLES

1-1	Summary of DMS-170 Components and	1-5
	Features	2-4
2-1	DML Statements	4-1
4-1	Error and Status Processing Mechanisms	4-3
4-2	Status Block Content	4-7
4-3	Locking Operations	6-1
6-1	Load Map Switch Settings	

NOTATIONS

The specifications for FORTRAN DML statements and for particular control statements are described in reference formats. The notations used in the reference formats are described as follows:

UPPERCASE Uppercase words are reserved words and must appear exactly as shown. You can use reserved words only as specified in the reference formats.

Lowercase Lowercase words are generic terms that represent user-supplied words or symbols.

[] Brackets enclose optional portions of a reference format. You can optionally omit or include all of the format within the brackets.

... Ellipses immediately follow a pair of brackets to indicate that you can optionally repeat the enclosed material.

Punctuation symbols shown within the formats are required unless enclosed in brackets and specifically noted as optional. One or more spaces separate the elements in a reference format. Numbers shown are decimal unless otherwise specified.

O

O

C

C

C

O

C

DMS-170 is a Control Data software package for data management. The system was designed on the premise that a data base should be centrally controlled and the data within that data base should be completely independent of application programs. In line with this philosophy, the role of data administrator emerged. This individual was to lead the design, programming, implementation, maintenance, and recovery efforts associated with the DMS-170 data management system.

The data administrator is responsible for the structural organization and layout of an entire data base. This individual assigns names to and describes the characteristics of all data items within the data base. This total description is called a schema. The schema is generated by the data administrator and stored as a permanent file.

As a FORTRAN programmer, you probably would never need or even want to access an entire data base. You would, however, need to access selected portions of a data base organized in a number of ways to meet the requirements of your various application programs. The grouping of data base items into separate data base portions is the responsibility of the data administrator. The descriptions of these grouped items are called sub-schemas. Sub-schemas are generated by the data administrator and stored in a permanent file library.

Internal control is handled by the CYBER Database Control System (CDCS). CDCS interprets all data base requests from application programs, ensures the validity of such requests, and passes them along to the input/output processor. The controls exercised by CDCS guarantee that one user cannot alter the contents of the data base and adversely affect another user's program.

The controls designated by the data administrator, incorporated into the schema and sub-schema, and carried out by CDCS relieve application programmers of many tedious tasks such as data description, data conversion, and validity checking.

SYSTEM COMPONENTS

The components of DMS-170 that are discussed in this guide include the language that describes the data (Data Description Language); the language that provides data base access to a FORTRAN application program (FORTRAN Data Manipulation Language); the module that controls data base activity (CYBER Database Control System); and the processor that handles all input and output operations (CYBER Record Manager). The components of DMS-170 that are not discussed in this guide include a special, nonprocedural language (Query Update) that provides data base access to programming and nonprogramming users and the COBOL Language extensions that provide data base access to COBOL application programs.

DATA DESCRIPTION LANGUAGE

The Data Description Language (DDL) is a compiler language that the data administrator uses to describe data. DDL can generate four types of descriptions: the schema definition that describes an entire data base; the FORTRAN sub-schema definition that describes selected portions of a schema-defined data base for use by a FORTRAN application program; the COBOL sub-schema definition that describes selected portions of a schema-defined data base for use by a COBOL application program; and the QUERY UPDATE sub-schema definition that either describes selected portions of a schema-defined data base or describes an independently controlled data base for use by the interactive query software product Query Update. The data descriptions for the schema and each sub-schema are declared in DDL source statements for input to the DDL compiler.

A block diagram illustrating schema/sub-schema generation is shown in figure 1-1.

The Schema

The schema is a detailed description of all the data in a data base. The schema description is generated from DDL statements that name the schema, organize the schema into files (called areas in the schema), describe each record type together with the characteristics of the data in the record, and describe relationships (called relations) and dependency conditions (called constraints) among areas. The schema also includes an access control capability that provides privacy at the area level.

The data administrator writes the DDL source statements and uses them as input to the DDL compiler for compilation into an object schema or schema directory. After storing the directory as a permanent file, the data administrator provides you with pertinent information so you can tailor your FORTRAN program to meet processing requirements. If, for example, you need to access an area that has been defined as having controlled access in the schema, it is the responsibility of the data administrator to supply you with the appropriate privacy key.

The Sub-Schema

The sub-schema is a detailed description of selected portions of the data in a data base. The FORTRAN sub-schema description is generated from DDL statements that identify the schema and sub-schema, specify files (called realms in the sub-schema) and the content and structure of records, indicate changes in data format required by the application program, identify relations to be used, and specify record qualification for relation processing.

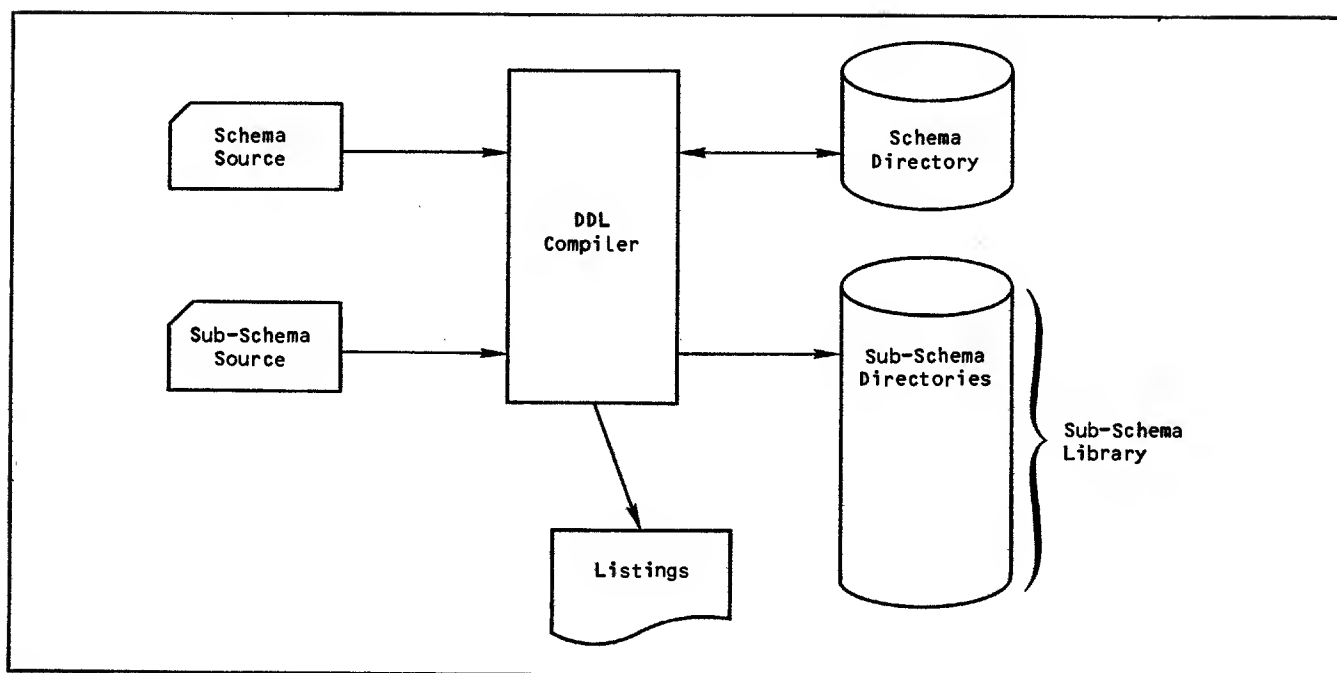


Figure 1-1. Schema and Sub-Schema Generation

The data administrator writes the DDL source statements and uses them as input to the DDL compiler for compilation into an object sub-schema or sub-schema directory. After storing the directory in the sub-schema library, the data administrator provides you with a listing of the sub-schema so you can obtain the names and descriptions of the data to be referenced in your FORTRAN program. The data administrator also provides you with the name of the sub-schema library, which you must attach with an operating system ATTACH control statement for DML preprocessing of the Data Manipulation Language statements in your FORTRAN program just before compilation.

FORTRAN DATA MANIPULATION LANGUAGE

The FORTRAN Data Manipulation Language (DML) is the language that provides a FORTRAN application program with access to the DMS-170 controlled data base. The language consists of a series of statements that provide for opening and closing of data base files; reading, writing, updating, and deleting records from those files; and relation processing. The DML statements you include in your FORTRAN source program code are translated by the DML preprocessor into statements acceptable to the FORTRAN compiler.

CYBER DATABASE CONTROL SYSTEM

The central controlling component of DMS-170 is CYBER Database Control System (CDCS), which monitors and interprets all data base requests from application programs. CDCS preprocesses each

application program request, performs any necessary data conversion, handles structural differences between the schema and the sub-schema by an operation called mapping, and prepares the request for input/output processing.

Master Directory

The master directory is a file that contains information relating to all data bases, schemas, and sub-schemas known to CDCS. The directory is generated by one of the data base utilities provided through CDCS. The data administrator creates the master directory and stores it as a permanent file. Your application program cannot reference a sub-schema unless information about that sub-schema exists in the master directory. It is the responsibility of the data administrator to ensure the sub-schema is valid. The master directory file is attached through the job stream of CDCS and is automatically available for your job.

CDCS Batch Test Facility

The CDCS Batch Test Facility is an absolute program that you can use during program development and testing. The facility enables you to run CDCS as a normal batch job, which means you can attach a new version of the master directory file each time you run a job.

The program, which resides on the system library, is called into execution by the CDCSBTF control statement. When using this facility, you are responsible for attaching the master directory file and any necessary log files each time you run a job.

Data Base Procedures

Data base procedures are special-purpose subprograms that CDCS calls when specific situations occur during CDCS processing. The data administrator writes the data base procedures and stores them in a permanent file library. The name of the procedure, the point at which it is to be called, and the conditions governing its execution are specified in the schema definition. Loading of data base procedures is handled automatically for you.

CYBER RECORD MANAGER

CYBER Record Manager (CRM) is the processor that performs all input/output operations for FORTRAN as well as the other CYBER host languages operating within DMS-170. The Advanced Access Methods (AAM) file manager handles all operations concerning the physical storage and access of data by application programs. All data base files supported by CDCS are conventional CRM files.

All necessary information regarding the characteristics of a data base file is supplied to CRM at schema compilation time. The data administrator specifies appropriate parameters on FILE control statements that are included in the DDL source deck when the schema is created. In DMS-170, all communication with CRM is handled automatically for you.

A block diagram illustrating the CRM interface with CDCS and the data base is shown in figure 1-2.

File Organization

File organization information is stored in the schema directory. The three file organizations

allowed for data base files that are to be accessed through CDCS are: indexed sequential, direct access, and actual key.

Records in indexed sequential files are stored in ascending order by key. An application program can access the records either randomly by key or sequentially.

Records in direct access files are stored randomly in fixed-length blocks. The number of the block to receive a record is determined by a calculation performed by the system on the record. An application program can access the records either randomly by key or sequentially.

Records in actual key files have key values assigned by the system. The key value is a number that identifies the block and the position within the block in which the record is stored. An application program can access the records either randomly by actual key or sequentially.

The primary key is specified in the schema. A listing of the sub-schema provides you with this information.

Multiple-Index Processing

Multiple-index processing is performed when alternate keys are defined for a file. An index is created for each alternate key in a data file when the file is created. The indexes are updated automatically whenever the data file is updated. An application program can retrieve the records by the primary key or by an alternate key.

Each alternate key is specified in the schema. A listing of the sub-schema provides you with this information.

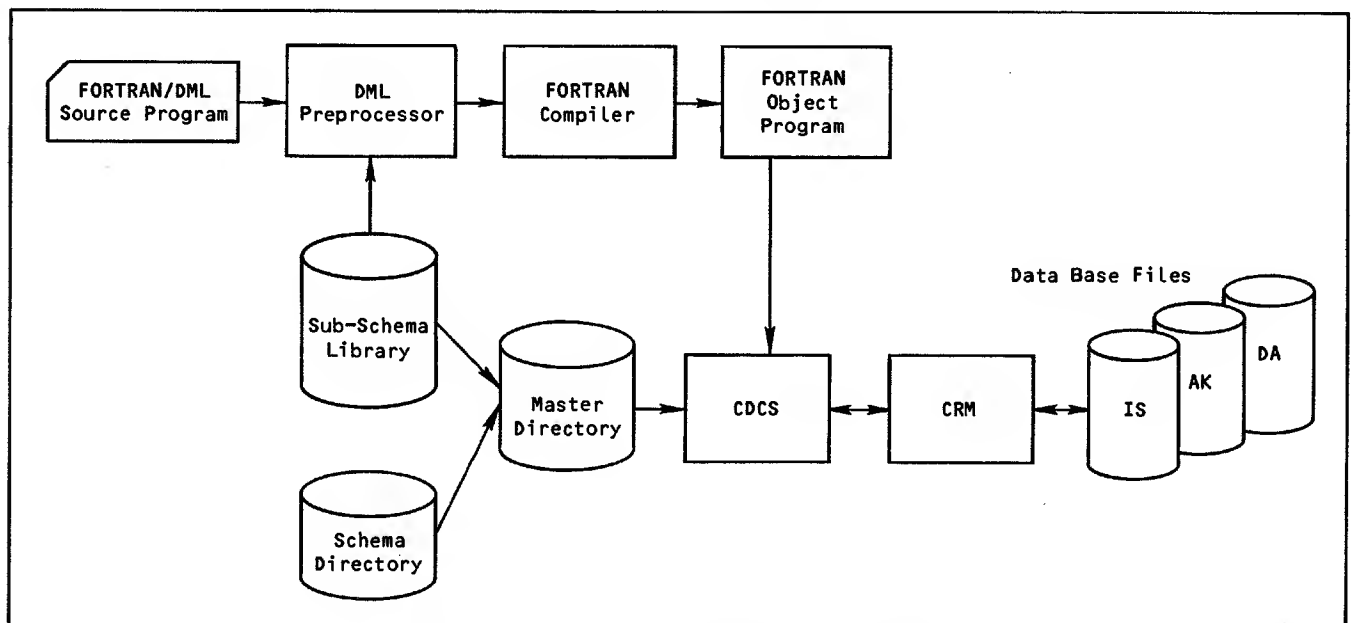


Figure 1-2. CYBER Record Manager Interface

SPECIAL FEATURES

Five special features with which you need to be familiar are: concurrency, file privacy, relations, constraints, and recovery. When these mechanisms are present in the CDCS operating environment, some action on the part of your application program might be required.

CONCURRENCY

The concurrency feature allows two or more application programs to access the same data base file at the same time for retrieval or update purposes. During concurrent update operations, CDCS provides a locking mechanism by which files and records can be locked and unlocked at appropriate times.

CDCS always locks the current record whenever the file is opened for input/output. Your application program, however, can issue explicit lock and unlock requests for CDCS to lock the entire file. By issuing a lock request, your program prevents other jobs from updating the file that it is using until it issues an unlock request. The file being locked and unlocked must be a file identified in the sub-schema.

A deadlock situation can occur when a program attempts to access files or records that have been locked by CDCS for other programs. When this situation occurs, CDCS arbitrarily releases the locked resources held by one of the contending programs. To ensure proper recovery handling in this type of situation, you should include appropriate code in your FORTRAN program.

FILE PRIVACY

The file privacy feature provides file access control. When file privacy has been specified in the schema, your program must supply privacy keys to gain access to the file.

The data administrator provides you with this information so you can ensure your FORTRAN program meets the privacy requirements when CDCS checks for appropriate privacy keys.

RELATIONS

The relational data base feature allows files to be linked together into a logical relationship called a relation. An application program can access the data from related files with a single read request. Relations are specified in the schema. Any relation that is available to an application program is specified in the sub-schema.

Your application program can access a relation by specifying a single read request with the name of the relation that is to be read. CDCS processes the request and returns a record occurrence from each file in the relation to your program's working storage area for the file.

The data administrator can place limitations on relations by including restrictions in the sub-schema. Restrictions are in the form of qualification criteria that must be satisfied before a record occurrence is made available to your program.

A listing of the sub-schema provides you with the name of the relation and indicates what specific restrictions apply.

CONSTRAINTS

The constraint feature allows controls to be imposed on update operations involving logically associated files. Constraints protect the integrity of the data base by allowing update operations to be performed only when specific conditions are satisfied. Constraints are specified in the schema and are enforced by CDCS.

The data administrator provides you with information concerning constraints. You can avoid constraint violations by becoming familiar with the rules that apply when modifying files on which constraints have been imposed.

RECOVERY

The recovery feature provides for reconstruction of a damaged or inconsistent data base and provides for the removing of updates made with erroneous logic. The data base can be recovered when physical storage or system failure occurs and all or part of the data base is lost or otherwise unreadable. The data base can be restored to a previous checkpoint or beginning of job when an application program failure or logic error occurs.

Recovery operations are made possible through a logging facility, which is the recording of user interactions with a data base file. Logging requirements are defined in the schema and serviced by CDCS. CDCS records the logging information on an independent file that ultimately serves as input for data base recover and restore operations. Log files, if specified, are attached through the job stream of CDCS and are automatically available to record the interactions of your program with the data base.

All logging specifications and recovery operations are performed under the jurisdiction of the data administrator. You can, however, define recovery points to CDCS in your FORTRAN program as an aid to recovery operations.

SUMMARY OF DMS-170 COMPONENTS AND FEATURES

A summary of DMS-170 components and features appears in table 1-1. This table provides a quick reference for appropriate information.

TABLE 1-1. SUMMARY OF DMS-170 COMPONENTS AND FEATURES

Component/ Feature	Definition	Information Appears In	Programmer Action
Alternate key	A key other than the primary key by which a file can be accessed; defined by the data administrator.	Sub-schema listing	On a random read, set by the application program to a value indicating the desired record occurrence.
CDCS Batch Test Facility	A simulator for use during program development.	N/A	Attach the master directory and appropriate log files when executing the application program.
Concurrency	Simultaneous access to the same data by two or more application programs during a given span of time.	N/A	Include appropriate code in the application program to handle a deadlock situation; deadlock can occur when two programs are contending for access to a locked file or record.
Constraints	Controls imposed on records in associated files or on items in a single file to protect the integrity of the data base during update operations; defined by the data administrator.	Schema	Obtain information from the data administrator. Follow the rules for modifying files on which constraints have been imposed.
CYBER Database Control System (CDCS)	The central controlling module of DMS-170.	N/A	None.
CYBER Record Manager (CRM)	The input/output processor for DMS-170 operations.	N/A	None.
Data base procedures	Special-purpose routines that perform predefined operations; written by the data administrator.	Schema	None.
Data Description Language (DDL)	The language that is used to structure a schema and sub-schema; used by the data administrator.	N/A	None.
File organization	The predetermined arrangement of stored data; indexed sequential, direct access, or actual key; defined by the data administrator.	Schema	None.
File privacy	A situation in which an application program can only gain access to a file by supplying a privacy key; defined by the data administrator.	Schema	Obtain information from the data administrator. Include a PRIVACY statement in the application program.
FORTTRAN Data Manipulation Language (DML)	The language that provides a FORTTRAN application program with access to the DMS-170 controlled data base.	N/A	Include appropriate DML statements in the FORTTRAN source program code.
Log files	Disk or tape files on which user interactions with data base files are recorded for recovery purposes.	Master directory	Attach the necessary log files only when executing the application program through the CDCS Batch Test Facility.
Master directory	A file containing information relating to all data bases, schemas, and sub-schemas known to CDCS; created by the data administrator.	N/A	Attach the master directory only when executing the program through the CDCS Batch Test Facility.

TABLE 1-1. SUMMARY OF DMS-170 COMPONENTS AND FEATURES (Contd)

Component/ Feature	Definition	Information Appears In	Programmer Action
Multiple-index processor	A processor that allows CRM files to be accessed by alternate keys.	N/A	None.
Primary key	A key that must be defined for a file when the file is first created; defined by the data administrator.	Sub-schema Listing	On a random read, set by the application program to a value indicating the desired record occurrence.
Recovery	A means by which a damaged or destroyed data base can be re-constructed; defined by the data administrator.	N/A	If practical, define a recovery point to CDCS in the application program.
Relations	Logical structures formed by the joining of files; permit retrieval of data from more than one file at the same time; defined by the data administrator.	Sub-schema Listing	To read a relation, specify a single read request with the name of the relation. During update, follow the rules for updating files joined in a relation.
Restrictions	Criteria that must be satisfied in a relation before a record occurrence can be made available to the application program; defined by the data administrator.	Sub-schema Listing	None.
Schema	A detailed description of all the data in a data base; created by the data administrator through DDL.	Schema Listing	None.
Sub-schema	A detailed description of selected portions of the data in a data base; created by the data administrator through DDL.	Sub-schema Listing	Attach the sub-schema library in which the sub-schema resides for DML preprocessing of the application program.

Every DMS-170 data base file that is to be accessed through FORTRAN must be described in a directory called a FORTRAN sub-schema. The data administrator, working with application programmers, is responsible for creating the sub-schemas. Every FORTRAN program that accesses a DMS-170 data base file must use the FORTRAN Data Manipulation Language (DML). The application programmer is responsible for coding appropriate DML statements and including them in the FORTRAN source program.

This section details the two principal data base access tools: the sub-schema that describes the data, and the language that provides access to that data.

INTERPRETING THE FORTRAN SUB-SCHEMA

The data administrator tailors sub-schemas to meet specific applications. Assume, for example, you have an application that requires access to only two fields in a data base file: student IDs and tuition charges. The data administrator might provide you with a sub-schema that resembles sub-schema SAMPLE1 shown in figure 2-1.

With the exception of the sub-schema library name and any required privacy keys, the listing provides you with complete information. The handwritten notation in this example indicates the sub-schema library name is DDLLIB. If you were planning to compile an application program using sub-schema SAMPLE1, you would need to attach DDLLIB. Since no privacy key is required, the schema obviously imposes no access control on realm ACCOUNT.

Notice the three aliases assigned. Since symbolic names in FORTRAN cannot exceed seven characters and cannot include a hyphen, the data administrator changes the names for your application.

Assume, for example, you have an application that requires access to two data base files. Assume, also, that you need a relationship between the two files so that you can search one file and retrieve corresponding records from the other. The data administrator might provide you with a sub-schema that resembles sub-schema SAMPLE2 in figure 2-2.

The handwritten notation in this example indicates the sub-schema library name is DDLLIB. If you were planning to compile an application program using sub-schema SAMPLE2, you would need to attach DDLLIB. The schema apparently imposes access control on realm FILE2. The privacy key XX99 must be included in a DML PRIVACY statement to gain access to that realm.

FORTRAN DATA MANIPULATION LANGUAGE

The FORTRAN Data Manipulation Language (DML) is the means through which your FORTRAN program accesses

the data base. You must code the DML statements along with FORTRAN statements in the FORTRAN source program. The DML statements identify the sub-schema, establish an interface with CDCS, and provide access to realms defined by the sub-schema. DML statements can appear both in the main program and in subprograms.

The DML statements are translated into statements acceptable to the FORTRAN compiler. The DML preprocessor performs the translation and writes the translated statements to a file along with the FORTRAN statements in the source program. This new file is then the input file to the FORTRAN compiler.

DML LANGUAGE COMPONENTS

The DML language components include DML statement keywords, recognized symbols and punctuation, and user-supplied names of variables and constants. These components are grouped together into statements for input to the DML preprocessor, which translates each statement appropriately into a FORTRAN specification or CALL statement.

The first word of a statement is always a DML keyword that identifies the task to be performed. Most keywords are followed by user-supplied elements and sometimes are followed by additional keywords.

A list of available DML statements is shown in table 2-1 for reference purposes. The statements are listed in alphabetic order by the leading word (keyword), which identifies the purpose of the complete statement. The comments column provides specific rules for the statement and includes applicable default options.

SYNTAX REQUIREMENTS

The syntax requirements and coding conventions for DML statements are exactly the same as for FORTRAN statements. The following restrictions apply:

- A DML statement cannot be the object of a logical IF.
- A DML statement must not reference files that are referenced elsewhere in the program by a conventional FORTRAN input/output statement or by a FORTRAN PROGRAM statement.

Any executable DML statement can have a statement label. The DML preprocessor copies the label into the translated FORTRAN statement.

STATEMENT POSITIONING

Some DML statements require special positioning within the FORTRAN source program. These requirements are illustrated in figure 2-3.

Figure 2-1. A Basic FORTRAN Sub-Schema

* SOURCE LISTING * (80351) DDLF

SUBSCHEMA SAMPLE2, SCHEMA=UNIVERSITY → The subschema name is SAMPLE2; the schema name is UNIVERSITY.

ALIAS (REALM) FILE1=STUDENT
 ALIAS (RECORD) FIL1REC=STUDENT-REC
 ALIAS (ITEM) FILE1ID=STUDENT-ID, STUDENT-REC
 → New names are assigned to two realms, records, and items. The new names are FILE1, FIL1REC, and FILE1ID, respectively, for the first realm; FILE2, FIL2REC, and FILE2ID, respectively, for the second realm.

ALIAS (REALM) FILE2=CURRICULUM
 ALIAS (RECORD) FIL2REC=CURR-REC
 ALIAS (ITEM) FILE2ID=STUDENT-ID, CURR-REC

REALM FILE1 }
 REALM FILE2 } → The sub-schema provides access to two realms: FILE1 and FILE2.

RECORD FIL1REC → The sub-schema provides access to record FIL1REC within FILE1.

CHARACTER*11 FILE1ID }
 CHARACTER*20 MAJOR } → Two items in record FIL1REC can be accessed: FILE1ID and MAJOR. FILE1ID is described as an 11-character item; MAJOR is described as a 20-character item.

RECORD FIL2REC → The sub-schema provides access to record FIL2REC within FILE2.

CHARACTER*14 IDENT
 CHARACTER*11 FILE2ID } → Three items in record FIL2REC can be accessed: IDENT, which is described as a 14-character item; FILE2ID, which is described as an 11-character item; and GRADE, which is described as a real item.

REAL GRADE

RELATION REL1 → A relation called REL1 is available. FILE1 and FILE2 are joined.

FILE1ID FOR AREA FILE1
 MAJOR FOR AREA FILE1
 IDENT FOR AREA FILE2
 FILE2ID FOR AREA FILE2
 GRADE FOR AREA FILE2
 → Primary and alternate keys are listed.

RECORD MAPPING IS NEEDED FOR REALM - FILE1
 RECORD MAPPING IS NEEDED FOR REALM - FILE2
 RESTRICT FIL2REC(GRADE .EQ. 4.0)
 END

END OF SUB-SCHEMA SOURCE INPUT

REL1 JOINS
 RELATION STATISTICS
 AREA - FILE1
 AREA - FILE2

 A restriction is placed on record FIL2REC. Only records with a grade point average of 4.0 are to be returned.

Library name DDLIB.
Privacy key 'XX99' needed for FILE2.

Figure 2-2. A Relational FORTRAN Sub-Schema

TABLE 2-1. DML STATEMENTS

Statement	Description	Comments
CLOSE	Ends processing of a realm.	Realms can be opened and closed any number of times by a program.
CLOSE relation	Ends processing of the realms joined in a relation.	Relations can be opened and closed any number of times by a program.
DELETE	Removes a record from a realm.	The record being deleted is the record most recently read from the realm. The value of the primary key cannot change after the last read.
INVOKE	Establishes the interface between the executing program and CDCS.	The statement must be executed before any other DML statements except SUBSCHEMA.
LOCK	Prevents other programs from updating a realm.	The realm must be a realm described in the sub-schema.
OPEN	Initiates processing of a realm.	If the processing mode is not specified, the realm is opened for input/output.
OPEN relation	Initiates processing of the realms joined in a relation.	If the processing mode is not specified, the relation is opened for input/output. A processing mode of open for output only is not valid.
PRIVACY	Establishes the right of a program to access a realm.	If the processing mode is not specified, the realm can be accessed for input/output. The mode must be the same as the mode indicated in the OPEN statement.
READ	Transfers data from a realm record to the variables defined in the sub-schema record description.	If a key is not specified, the read is sequential. If a key is specified, the value of the referenced key must be set by the program before the read is executed.
READ relation	Transfers data from the relation records to the corresponding variables defined in the sub-schema record descriptions.	If a key is not specified, the read is sequential. If a key is specified, the key must be in the root realm; the value of the referenced key must be set by the program before the read is executed.
REWRITE	Replaces the last record read with a new record, using the current values of the variables defined in the sub-schema record description.	The value of the primary key must not have changed since the last read.
START	Logically positions a realm for a subsequent sequential read operation.	The processing mode must be either input or input/output. If a key is specified, it must be a primary or alternate key defined for the realm. If a key is not specified, positioning is by primary key.
START relation	Logically positions the root realm (the first realm named in the sub-schema) of a relation for a subsequent relation read operation.	The processing mode must be either input or input/output. If a key is specified, it must be a primary or alternate key that is defined in the root realm. If a key is not specified, positioning is by primary key of the root realm.
SUBSCHEMA	Identifies the sub-schema to be used by the program.	A FORTRAN program can reference only one sub-schema.
TERMINATE	Terminates the interface between the FORTRAN program and CDCS.	Only another INVOKE statement can follow a TERMINATE statement.
UNLOCK	Releases a lock on a realm.	The realm must be a realm described in the sub-schema.
WRITE	Writes a record, using the current values of the variables defined in the sub-schema record description.	Schema record data items that are not defined in the sub-schema are given null values by CDCS.

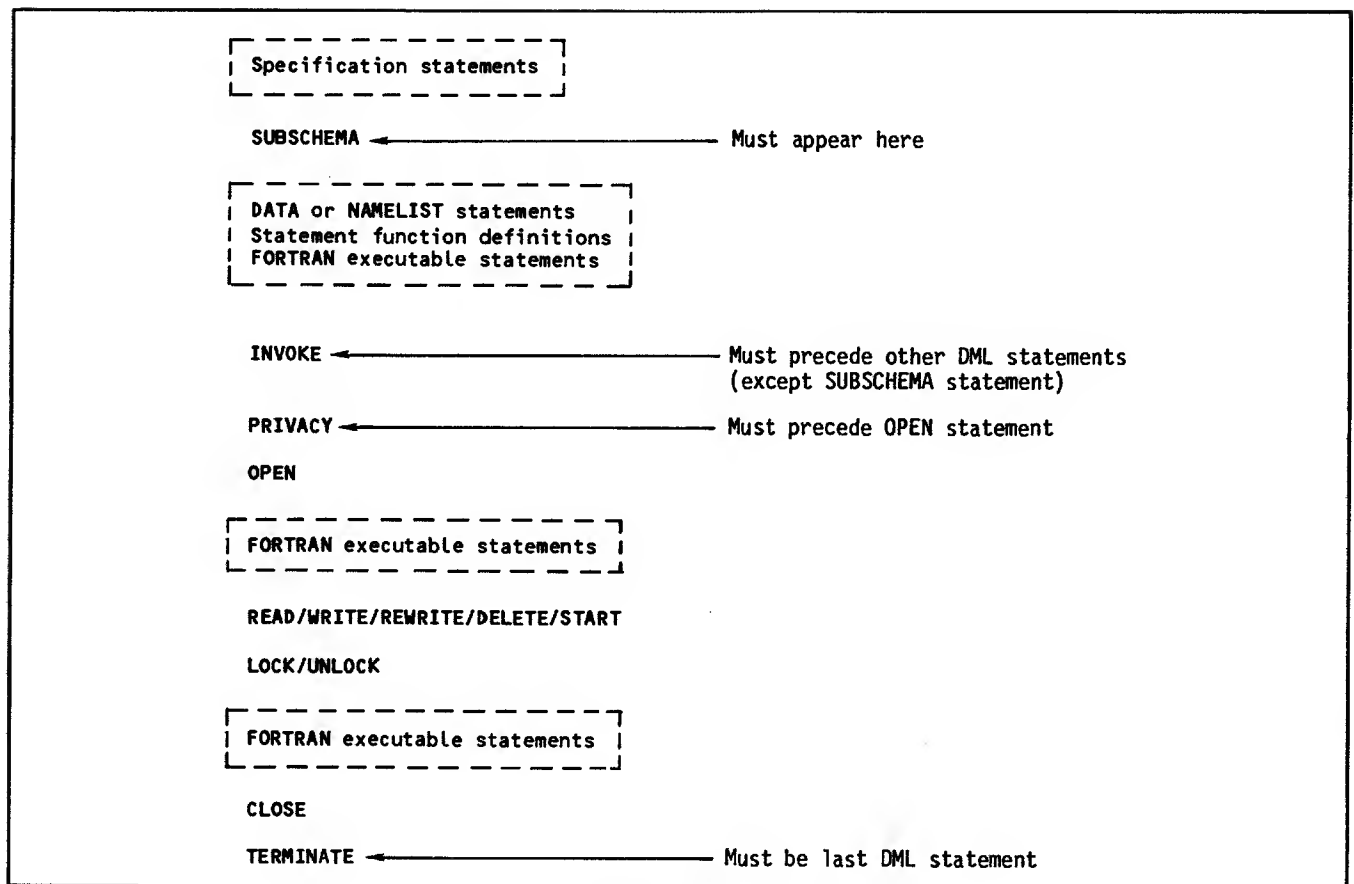


Figure 2-3. DML Statement Positioning

○

○

○

○

○

○

○

Processing data base files within the DMS-170 environment involves several steps. These steps are:

1. Obtain a current listing of the sub-schema from the data administrator so you can have the names and descriptions of the data your program will be referencing.
2. Obtain the name of the appropriate sub-schema library from the data administrator. You will need to attach this library for preprocessing your program.
3. Ask the data administrator if any realms in the sub-schema are defined in the schema as having controlled access. When access is controlled, you must know the privacy key.
4. Ask the data administrator if any constraints exist in the schema. When constraints exist, CDCS enforces them by not allowing updates that violate constraints.
5. Code the FORTRAN program and include appropriate Data Manipulation Language (DML) statements for opening, closing, and processing sub-schema realms.
6. Preprocess and compile the FORTRAN program. Include, in the job stream before the FTNS control statement, an ATTACH control statement naming the sub-schema library and a DML control statement to execute the DML preprocessor.
7. When compilation is successful, execute the FORTRAN program. Include an LDSET control statement to load the DMS-170 library.

DML statements are available to perform a variety of operations on data base items described in a FORTRAN sub-schema. This section describes these statements and presents them in the following sequence:

Data Base Access

SUBSCHEMA
INVOKE
PRIVACY
OPEN
LOCK/UNLOCK
CLOSE
TERMINATE

Data Base Manipulation

WRITE
READ
START
REWRITE
DELETE

Relation Access

OPEN
CLOSE
READ
START

For purposes of illustration, a new sub-schema named AVERAGE is shown in figure 3-1. This sub-schema is referenced in subsequent examples. The examples show portions of program MODEL which illustrate statements necessary for particular data base operations.

The sub-schema provides the following information:

- The realm (file) to be accessed is named CFILE.
- The record is named CRECORD.
- A character item named IDENT is the primary key.
- A character item named STUDENT is an alternate key.
- A character item named COURSE is an alternate key.
- A real item named GRADE is an alternate key.

The handwritten notation on the listing indicates the sub-schema is stored on a library named SSLIB. The notation also indicates CFILE has controlled access and requires a privacy key of XX99.

USING DML TO ACCESS THE DATA BASE

To access a data base, a FORTRAN program must identify the sub-schema that the program uses, establish an interface with CDCS, satisfy privacy requirements, and perform the usual functions of opening and closing files. The following paragraphs describe these functions and the DML statements that you include in your program to provide these functions.

IDENTIFYING THE SUB-SCHEMA

To identify the sub-schema, you must include a SUBSCHEMA statement in your program. This must be the first DML statement to appear in your program. The format is:

SUBSCHEMA(sub-schema-name)

The SUBSCHEMA statement is required. You must position the statement in the program as follows:

- After the specification statements
- Before the first DATA or NAMELIST statement

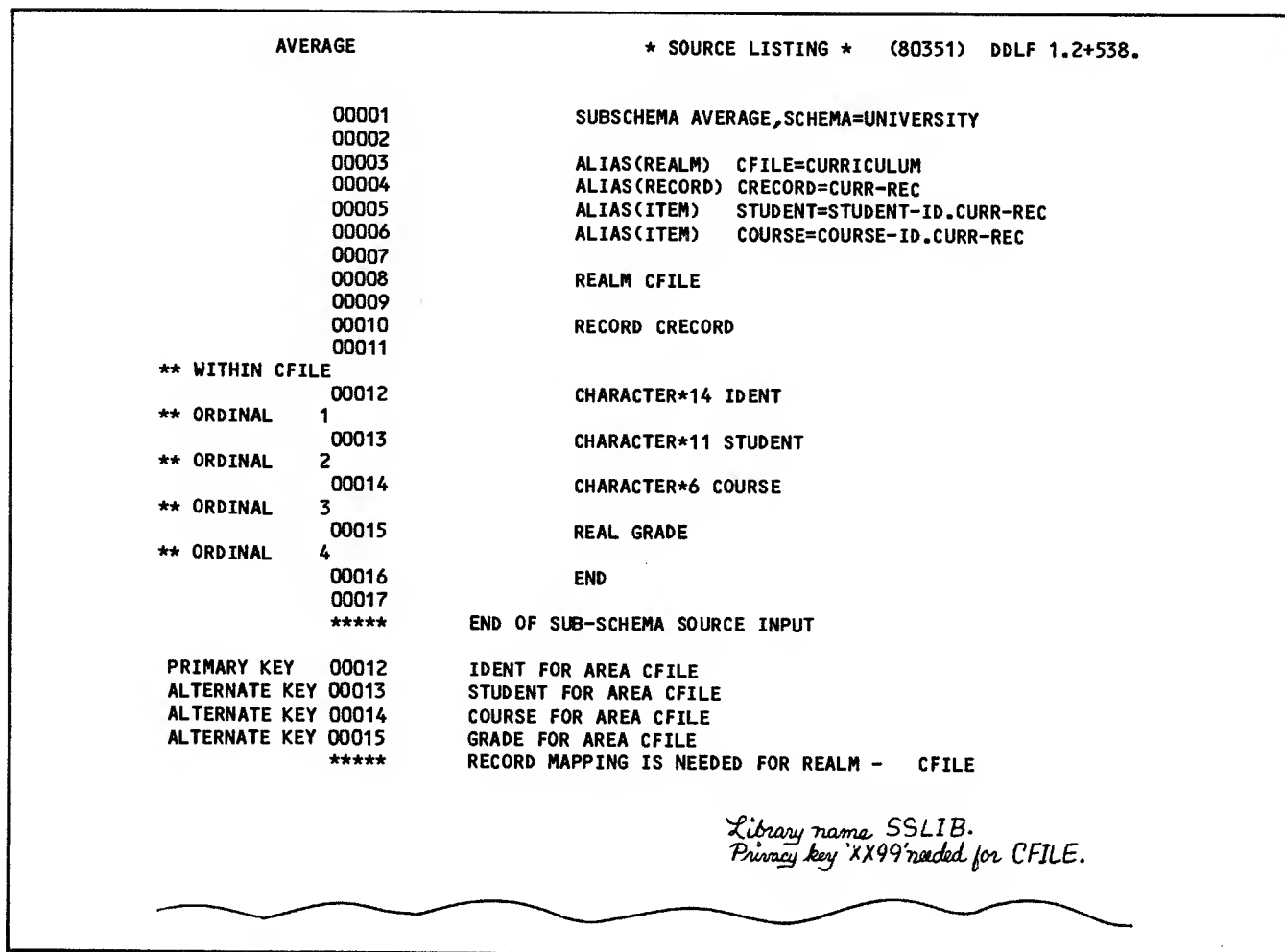


Figure 3-1. Sub-Schema AVERAGE

- Before any statement function
- Before any executable statement

At the point where the DML preprocessor encounters the SUBSCHEMA statement, the DML preprocessor copies into the source program the text declaration and DATA statements resulting from the sub-schema compilation. In this way, DML provides your program with the ability to reference all records, data items, and relations that are described in the sub-schema.

In a program using the sample sub-schema AVERAGE, the SUBSCHEMA statement appears as shown in figure 3-2.

```

PROGRAM MODEL
.
.
.
CHARACTER ...
DIMENSION ...
.
.
.
SUBSCHEMA(AVERAGE)
DATA ...
DO ...
.
.
.
END

```

Figure 3-2. Identifying the Sub-Schema

The INVOKE statement is required. The statement must appear in the program before any other DML statement except SUBSCHEMA.

When the INVOKE statement is executed, CDCS automatically attaches for use by the program all realms described in the sub-schema identified by the program.

ESTABLISHING THE INTERFACE WITH CDCS

To establish the interface with CDCS, you must include an INVOKE statement in your program. This must be the second DML statement to appear in your program. The format is:

INVOKE

In a program using the sample sub-schema AVERAGE, the INVOKE statement appears as shown in figure 3-3.

```

PROGRAM MODEL
.
.
.
CHARACTER ...
DIMENSION ...
.
.
.
SUBSCHEMA(AVERAGE)
DATA ...
DO ...
.
.
.
INVOKE
.
.
.
END

```

Figure 3-3. Establishing the Interface With CDCS

SATISFYING PRIVACY REQUIREMENTS

If a realm is defined in the schema as having controlled access, your program must provide a privacy key to access the realm. To provide the privacy key, you must include the PRIVACY statement in your program. The format is:

```

PRIVACY(realm-name, [MODE=mode,]
        PRIVACY=privacy key)

```

where

```

mode =      I (access allowed for input)

           IO (access allowed for both
              input and output, called
              input/output; default)

           O (access allowed for output)

privacy key = character constant, variable
              name, unsubscripted array name

```

A privacy key can be from 1 through 30 characters in length. If you use a character constant to specify a privacy key, enclose the character string in quotes. If you use a variable name to specify a privacy key, define the variable as type CHARACTER*30. Ensure that the privacy key is left-justified and blank filled in the field of the variable. If you use an array name to specify the privacy key, define the array as a 3-word array. Ensure that the privacy key is left-justified and blank filled in the field of the array.

The PRIVACY statement is required when access is controlled. A separate PRIVACY statement is required for each realm defined with controlled access. The PRIVACY statement must be executed before the statement that opens the realm.

The handwritten notation on the sample sub-schema listing (figure 3-1) indicates access to the realm is controlled and a privacy key called XX99 is required. In a program using the sample sub-schema AVERAGE, the required PRIVACY statement appears as shown in figure 3-4.

```

PROGRAM MODEL
.
.
.
SUBSCHEMA(AVERAGE)
.
.
.
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
.
.
.
END

```

Figure 3-4. Satisfying Privacy Requirements

OPENING A REALM

Before your program can access any data records in an existing realm, the program must open the realm. To open the realm, you must include the OPEN statement in your program. The format is:

```

OPEN(realm-name [,MODE=mode] [,ERR=s])

```

where

```

mode = I (open for input only)

       IO (open for both input and output,
          called input/output; default)

       O (open for output only; valid only for
          creating a new file)

s =    label of an executable statement to
       which control transfers on open error

```

For a realm with controlled access, the mode of access indicated in the OPEN statement must be the same as the mode indicated in the PRIVACY statement.

In a program using the sample sub-schema AVERAGE (figure 3-1), the OPEN statement appears as shown in figure 3-5. The MODE option is not included, indicating a default to input/output. Both the PRIVACY and OPEN statements indicate the same mode, input/output. If an error occurs on open, program execution continues at statement 100.

LOCKING/UNLOCKING A REALM

Whenever your program issues a read request on a realm that is open for input/output, CDCS automatically locks the record that was read (the current record) against update by another user. Through DML, however, your program can prevent other jobs from performing update operations

```

PROGRAM MODEL
.
.
.
SUBSCHEMA(AVERAGE)
.
.
.
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
.
OPEN(CFILE,ERR=100)
.
.
.
100 PRINT *, 'ERROR ON OPEN'
.
.
.
END

```

Figure 3-5. Opening a Realm

anywhere within a realm by issuing a request that CDCS lock the entire realm. To issue the lock request, you must include the LOCK statement in your program. The format is:

LOCK(realname [,ERR=s])

where

s = label of an executable statement to which control transfers on lock error

The LOCK statement should be executed before any read request with intent to update the record.

CDCS releases the realm lock held for your program when the program issues an unlock request. To issue the unlock request, you must include an UNLOCK statement in your program. The format is:

UNLOCK(realm-name [,ERR=s])

where

s = label of an executable statement to which control transfers on unlock error

You should judiciously use a realm lock. A realm lock limits other users' access to the realm (file); other application programs can read but cannot update the file. Additionally, a realm lock overrides the CDCS record locking mechanism, which provides a checking capability on rewriting and deleting records (for additional information, see the paragraphs *Rewriting a Record* and *Deleting a Record*).

In a program using the sample sub-schema AVERAGE (figure 3-1), the LOCK and UNLOCK statements appear as shown in figure 3-6. If an error occurs during the lock or unlock process, program execution continues at statement 200 or 300, respectively.

```

PROGRAM MODEL
.
.
.
SUBSCHEMA(AVERAGE)
.
.
.
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
.
OPEN(CFILE,ERR=100)
.
.
LOCK(CFILE,ERR=200)
.
.
.
UNLOCK(CFILE,ERR=300)
.
.
.
200 PRINT *, 'ERROR ON LOCK'
.
.
.
300 PRINT *, 'ERROR ON UNLOCK'
.
.
.
END

```

Figure 3-6. Locking/UnLocking a Realm

CLOSING A REALM

When your program has completed processing on a realm, the program must close the realm. To close the realm, you must include the CLOSE statement in your program. The format is:

CLOSE(realname [,ERR=s])

where

s = label of an executable statement to which control transfers on close error

Once a program closes a realm, the program can perform no further processing on the realm until it reopens the realm.

In a program using the sample sub-schema AVERAGE (figure 3-1), the CLOSE statement appears as shown in figure 3-7. If an error occurs on close, program execution continues at statement 400.

TERMINATING THE INTERFACE WITH CDCS

To terminate the interface with CDCS, you must include a **TERMINATE** statement in your program. The format is:

TERMINATE

Once the **TERMINATE** statement is executed, no further data base processing can take place without execution of another **INVOKE** statement.

In a program using the sample sub-schema AVERAGE (figure 3-1), the TERMINATE statement appears as shown in figure 3-8. This statement must be executed before the FORTRAN END or STOP statement.

```

PROGRAM MODEL
.
.
.
SUBSCHEMA(AVERAGE)
.
.
.
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
.
OPEN(CFILE,ERR=100)
.
.
LOCK(CFILE,ERR=200)
.
.
.
UNLOCK(CFILE,ERR=300)
.
.
CLOSE(CFILE,ERR=400)
.
.
400 PRINT *, 'ERROR ON CLOSE'
.
.
.
END

```

Figure 3-7. Closing a Realm

```

PROGRAM MODEL
.
.
.
SUBSCHEMA(AVERAGE)
.
.
.
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
.
OPEN(CFILE,ERR=100)
.
.
.
CLOSE(CFILE,ERR=400)
.
.
100 PRINT *, 'ERROR ON OPEN'
.
.
.
400 PRINT *, 'ERROR ON CLOSE'
.
.
.
TERMINATE
.
.
END

```

Figure 3-8. Terminating the Interface With CDCS

USING DML TO MANIPULATE DATA

DML statements are available to create, read, position, and modify data base records. The following paragraphs describe these functions and the DML statements that you must include in your program to provide these functions.

WRITING A RECORD

To write a complete record, you must include a WRITE statement in your program. The format is:

```
WRITE(realname [,ERR=s])
```

where

s = label of an executable statement to which control transfers on write error

Before the WRITE statement is executed, the program must set the primary key and all alternate keys to appropriate values. A sub-schema does not always reflect all data items that appear in the schema record; therefore, before allowing the new record to be written to the data base, CDCS gives null values to those schema data items that are not defined in the sub-schema.

In a program using the sample sub-schema AVERAGE (figure 3-1), the WRITE statement appears as shown in figure 3-9. The program sets the primary key IDENT and alternate keys STUDENT, COURSE, and GRADE to appropriate values before the WRITE statement is executed. If an error occurs on write, program execution continues at statement 500.

```

PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,MODE=IO,ERR=100)
IDENT='122-13-6704-09'
STUDENT='122-13-6704'
COURSE='PSY136'
GRADE=3.7
WRITE(CFILE,ERR=500)
.
.
.
500 PRINT *, 'ERROR ON WRITE'
.
.
.
900 CLOSE(CFILE)
TERMINATE
END

```

Figure 3-9. Writing a Record

This example shows a program writing a record to an existing file, CFILE. Before the program writes to this file, the PRIVACY and OPEN statements establish access for input/output (MODE=IO). If the program were creating this file, the OPEN and PRIVACY statements would have to establish access for output only (MODE=O).

READING A RECORD

To read a record, you must include a READ statement in your program. The format is:

```
READ(real-name [,KEY symbol item-name]
      [,ERR=s] [,END=s])
```

where

symbol = = .EQ. .GT. .GE.

item-name = primary or alternate key

s = Label of an executable statement to
 which control transfers on read
 error (ERR)

 Label of an executable statement to
 which control transfers on
 end-of-file (END)

When you omit the KEY option, the read operation is sequential. When you include the KEY option, the read operation is random.

The END option is valid only for a sequential read operation.

Sequential Read

A sequential read accesses the record occurrence located at the current record position. Successive read operations return record occurrences by position. Indexed sequential files are sequenced in ascending primary key order, actual key files are sequenced by block and record slot within the block, and direct access files are sequenced by position in home blocks.

Typical FORTRAN 5 statements issued outside of a data base environment terminate with a fatal error if EOF is sensed and a test for EOF status is not included in the FORTRAN READ statement. If EOF status is not tested in DML, program execution continues with the next statement. Consequently, it is necessary to test for EOF on a DML sequential read operation. You can handle this test in one of two ways:

- Include the END option on the READ statement. When EOF is sensed, program execution continues at the statement specified in the option.
- Test for an EOF value of 100g in the data base status block. This option is described in section 4.

In a program using the sample sub-schema AVERAGE (figure 3-1), a sequential read appears as shown in figure 3-10. If an error occurs on read, program execution continues at statement 600. The READ statement includes the END option to test for EOF. When EOF is reached, program execution continues at statement 900.

Random Read

A random read accesses a record occurrence by the value of a referenced primary or alternate key. The program must set the value of the referenced key before the READ statement is executed.

```
PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
READ(CFILE,ERR=600,END=900)
.
.
.
600 PRINT *, 'ERROR ON READ'
.
.
.
900 CLOSE(CFILE)
TERMINATE
END
```

Figure 3-10. Reading Sequentially

In a program using the sample sub-schema AVERAGE (figure 3-1), a random read appears as shown in figure 3-11. The program sets the alternate key GRADE to the value 4.0. This read returns from CFILE the first record occurrence in which the alternate key GRADE has the value 4.0. If an error occurs on read, program execution continues at statement 600.

```
PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
GRADE=4.0
READ(CFILE,KEY.EQ.GRADE,ERR=600)
.
.
.
600 PRINT *, 'ERROR ON READ'
.
.
.
900 CLOSE(CFILE)
TERMINATE
END
```

Figure 3-11. Reading Randomly

POSITIONING A REALM

To position a realm for subsequent sequential read operations, you must include a START statement in your program. The format is:

```
START(real-name [,KEY symbol item-name]
      [,ERR=s])
```

where

symbol = = .EQ. .GT. .GE.

item-name = primary or alternate key

s = Label of an executable statement to
 which control transfers on start
 error

Before the START statement is executed, the program must have opened the realm for input or input/output.

When you omit the KEY option, the realm is positioned by primary key value; the realm is positioned to the record occurrence with a primary key value equal to the current value of the primary key item. When you include the KEY option, the realm is positioned to the first record occurrence with a matching key value.

In a program using the sample sub-schema AVERAGE (figure 3-1), both forms of the START statement appear as shown in figure 3-12. If an error occurs on start, program execution continues at statement 700.

```

PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
IDENT='122-13-6704-01'
START(CFILE,ERR=700)
READ(CFILE,ERR=600,END=900)
.
.
.
700 PRINT *, 'ERROR ON START'
.
.
.
COURSE='PSY100'
START(CFILE,KEY.GE.COURSE,ERR=700)
READ(CFILE,ERR=600,END=900)
.
.
.
900 CLOSE(CFILE)
TERMINATE
END

```

Figure 3-12. Positioning a Realm

The first START statement omits the KEY option, which means CFILE will be positioned by primary key. The program sets the primary key (IDENT) to the value 122-13-6704-01 before the START statement is executed; the realm will be positioned to the record occurrence with the matching primary key value.

The second START statement includes the KEY option. The program sets the alternate key COURSE to the value PSY100. The first record occurrence in CFILE with an alternate key COURSE greater than or equal to PSY100 will be the one to which CFILE is positioned.

REWRITING A RECORD

To rewrite a record, you must include a REWRITE statement in your program. The format is:

```
REWRITE(realm-name [,ERR=s])
```

where

s = label of an executable statement to which control transfers on rewrite error

When a record is rewritten, the current values of those variables defined in the sub-schema are rewritten to the specified data base record. Data items defined in the schema but not defined in the sub-schema remain unchanged.

Before your program can rewrite a record, your program must have locked the record either with a record lock or with a realm lock. Typically, the record lock is used.

For a rewrite using a record lock, the program establishes the record locking mechanism by opening the realm for input/output. To rewrite the record, the program must include the following steps:

1. Read the record to the program's working storage area by executing a DML READ statement.
2. Set the value of each data item being changed to the appropriate new value.
3. Rewrite the record by executing a REWRITE statement.

When the realm is opened for input/output and the read is executed, CDCS expects an update operation and consequently locks the record. CDCS allows rewriting of only the locked record.

The program must not change the value of the primary key between the read and the rewrite of the record. The following example illustrates a processing sequence to avoid:

```

IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT)
IDENT='200-44-7863-01'
REWRITE(CFILE)

```

Assuming a rewrite using a record lock, CDCS does not allow the rewrite in this example to be performed because the record is not locked; record 100-22-5860-04 is locked, but the rewrite is attempted on record 200-44-7863-01. CDCS issues an error diagnostic on the rewrite.

If an update requires that the value of a primary key be changed, the program must first delete the record with the old primary key value and then write the record with the new primary key value.

For a rewrite using a realm lock, the program establishes the realm lock by executing the LOCK statement. Then to rewrite a record, the program needs only to set the value of the primary key to the value of the record being rewritten and execute the REWRITE statement. The recommended rewriting procedure, however, includes more steps than these. The recommended procedure is the same as for a rewrite using a record lock: read the record, change the appropriate values, then rewrite the record. By reading the record, the program can test for an error on the read and, thereby, protect the integrity of the data base. With the realm lock, it is your responsibility to ensure that the program does not change the value of the primary key between the read and the rewrite of the record.

You should judiciously use a realm lock when rewriting records because the realm lock overrides the record lock and the checking capability available through the record lock.

In a program using the sample sub-schema AVERAGE (figure 3-1), the REWRITE statement appears as shown in figure 3-13. The program performs the rewrite by using a record lock. The program reads the record occurrence with a primary key value of 100-22-5860-04, changes the value of data item GRADE to the value 3.8, and then rewrites the record. In the rewritten record, all other values in the record occurrence remain unchanged. If an error occurs on rewrite, program execution continues at statement 800.

```

PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT,ERR=600)
GRADE=3.8
REWRITE(CFILE,ERR=800)
.
.
.
600 PRINT *, 'ERROR ON READ'
.
.
.
800 PRINT *, 'ERROR ON REWRITE'
.
.
.
900 CLOSE(CFILE)
TERMINATE
END

```

Figure 3-13. Rewriting a Record

DELETING A RECORD

To delete a record, you must include a DELETE statement in your program. The format is:

```

DELETE(realn-name [,ERR=s])
where
    s = label of an executable statement to which
        control transfers on delete error

```

Before your program can delete a record, your program must have locked the record either with a record lock or with a realm lock. Typically, the record lock is used.

For a delete using a record lock, the program establishes the record locking mechanism by opening the realm for input/output. To delete the record, the program must include the following steps:

1. Read the record by executing a DML READ statement.
2. Delete the record by executing a DELETE statement.

When a realm is opened for input/output and the read is executed, CDCS expects an update operation and consequently locks the record. CDCS allows deletion of only the locked record.

The program must not change the value of the primary key between the read and the delete of the record. The following example illustrates a processing sequence to avoid:

```

IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT)
IDENT='400-23-1248-07'
DELETE(CFILE)

```

Assuming a delete using a record lock, CDCS does not allow the delete in this example to be performed because the record is not locked; record 100-22-5860-04 is locked, but the delete is attempted on record 400-23-1248-07. CDCS issues an error diagnostic on the delete.

For a delete using a realm lock, the program establishes the realm locking mechanism by executing the LOCK statement. Then to delete a record, the program needs only to set the value of the primary key to the value of the record being deleted and execute the DELETE statement. The recommended procedure for deleting a record, however, includes more steps than these. The recommended procedure is the same as for a delete using a record lock: read the record and then delete the record. By reading the record, the program can test for an error on the read and, thereby, protect the integrity of the data base. With the realm lock, it is your responsibility to ensure that the program does not change the value of the primary key between the read and the delete of the record.

You should judiciously use a realm lock when deleting records because the realm lock overrides the record lock and the checking capability available through the record lock.

In a program using the sample sub-schema AVERAGE (figure 3-1), the DELETE statement appears as shown in figure 3-14. The program performs the delete by using a record lock. The program reads from CFILE the record occurrence with a primary key (IDENT) equal to 100-22-5860-04 and then deletes that record. If an error occurs on delete, program execution continues at statement 850.

USING DML TO PROCESS RELATIONS

Relation processing greatly simplifies programming when several related realms are required by the application program. Realms that have common data items can be joined in a relation. When a relation is included in a sub-schema, the relation can be accessed and read through DML. This means that a single relation read request by an application program returns a relation occurrence, which consists of one qualifying record from each of the realms comprising the relation.

```

PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKE
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT,ERR=600)
DELETE(CFILE,ERR=850)
.
.
.
600 PRINT *, 'ERROR ON READ'
.
.
.
850 PRINT *, 'ERROR ON DELETE'
.
.
.
900 CLOSE(CFILE)
TERMINATE
END

```

Figure 3-14. Deleting a Record

DML statements are available to provide for processing relations. The functions involved in relation processing are opening and closing the realms of the relation, positioning a relation through a start operation, and reading the relation. Paragraphs that follow describe these functions and the DML statements that you must include in your program to provide the functions. First, however, it is necessary to examine the structure of a relation and the manner in which CDCS returns records to the program's working storage area.

STRUCTURE OF A RELATION

A relation can be described as a hierarchical tree structure. The root of the tree is the realm through which the relation is entered; this is the first realm listed for a relation in the sub-schema. A data item in the realm at the root of the structure joins the realm to a common data

item in the next realm listed for the relation. When the relation is entered, the value of the data item in the root realm record leads to a record in the second realm. More than one record in the second realm can contain the same value; thus one record in the root realm can lead to several records in the second realm.

The second realm in the relation can be joined to a third realm through a common data item. Once again, a record in the second realm can lead to several records in the third realm. This branching out from the root of the tree continues through each realm in the relation.

The tree structure of the three-realm relation REL3 of the university data base is illustrated in figure 3-15. The tree structure is normally pictured upside down, with the root at the top and branches going down. The first realm, PFILE, which is the root of the structure, consists of a master record for each professor. The second realm, CRSFILE, consists of a master record for each course. The third realm, CFILE, consists of curriculum records. The common data item joining the first and second realms is the professor identification; the common data item joining the second and third realms is the course identification.

Realms in a relation are numbered consecutively as ranks. The first realm entered (called the root realm) is always assigned rank 1. The rank is incremented by 1 for each successive realm in the relation. The value of the rank of a realm contrasts with the placement of the realm in the tree structure. The lower the rank, the higher the realm is shown in the tree structure; i.e., rank 1 (the lowest rank) is shown at the top of the tree structure. Figure 3-15 also shows ranks in the relation REL3.

When a relation is read, a record occurrence from each realm in the relation is returned to the program. A relationship exists between record occurrences in a relation: a parent/child relationship. A record occurrence that has another record occurrence at the next numerically higher rank in the relation is referred to as the parent record

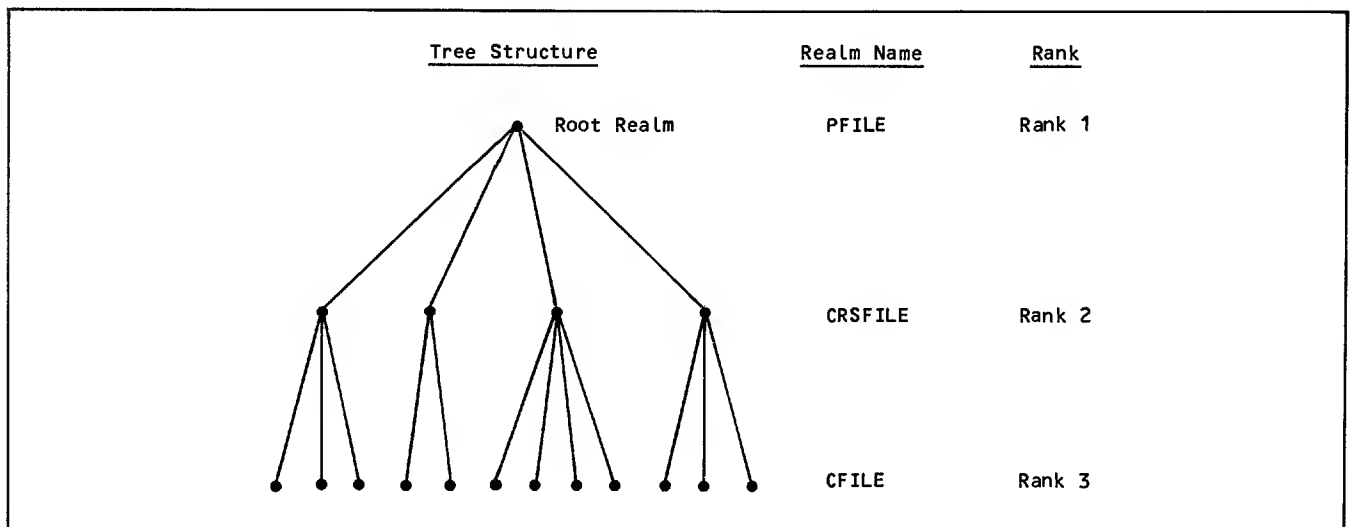


Figure 3-15. Tree Structure and Ranks of a Three-Realm Relation

occurrence. A record occurrence that has another record occurrence at the next numerically lower rank in the relation is referred to as the child record occurrence. In a parent/child relationship in relation REL3, a record occurrence in PFILE would represent the parent with corresponding record occurrences of CRSFILE representing the children. Additionally, a record occurrence in CRSFILE would represent the parent with corresponding record occurrences in CFILE representing the children.

USING THE SUB-SCHEMA

The structure of a relation is defined when the schema is created. A relation that is available to an application program is included in the sub-schema. The sub-schema listing provides the names of the realms in the relation. A new sample sub-schema named COMPARE, which makes available a three-realm relation, is shown in figure 3-16.

The sub-schema listing provides the following information:

- A relation is available to DML because the **RELATION** statement is included in the sub-schema.
- Three schema areas are joined by relation named REL3. The areas are named PROFESSOR, COURSE, and CURRICULUM in the schema; they are renamed as realms PFILE, CRSFILE, and CFILE in the sub-schema. The order in which the areas (realms) appear in the Relation Statistics portion of the listing indicates the ranks of the realms: the first realm listed has the rank 1; the second, rank 2; and so forth.
- PFILE has a primary key named PROFID; CRSFILE has an alternate key named PROF. Looking at the listing of aliases, you can see these fields both appear as PROF-ID in the schema. Obviously these fields represent unique professor identification and are common to both realms. You can assume that these items join the realms. The data administrator, however, should provide the common data items if they are not obvious and if programming considerations require that you know them.
- CRSFILE has a primary key named CRSID; CFILE has an alternate key named COURSE. Looking at the listing of aliases, you can see these fields both appear as COURSE-ID in the schema. Obviously these fields represent unique course information and are common to both realms. You can assume that these items join the realms. The data administrator, however, should provide the joining data items under the conditions indicated previously.
- A restriction is placed on CRECORD. A relation occurrence will not be returned unless data item CODE contains the character C. Before a relation occurrence is returned to the program's working storage area, CDCS checks for restrictions and enforces any restrictions. CDCS allows only qualifying records to be returned.

OPENING A RELATION

Before your program can access any data records in an existing relation, the program must open the appropriate realms. To open all the realms in a relation, you can include a relation OPEN statement in your program. The format is:

```
OPEN(relation-name [,MODE=mode] [,ERR=s])
```

where

mode = I (open for input only)

IO (open for both input and output, called input/output; default)

s = label of an executable statement to which control transfers on open error

Your program should normally open a relation for input (MODE=I). The program should open the relation for input/output (MODE=IO) under two circumstances:

- Processing requirements indicate that the program should lock the records to prevent update during the relation read.
- The program updates individual realms in the relation following the relation read.

If your program is opening a relation in which one or more realms have controlled access, you must include in the program a PRIVACY statement for each realm that has controlled access.

The following statement opens for input the realms joined in relation REL3, which is shown in sample sub-schema COMPARE (figure 3-16). If an error occurs on open, program execution continues at statement 50:

```
OPEN(REL3,MODE=I,ERR=50)
```

If you have included an OPEN statement in your program for each realm in the relation, you do not need to include a relation OPEN statement.

CLOSING A RELATION

When your program has completed relation processing, the program must close the appropriate realms. To close all the realms of a relation, you can include a relation CLOSE statement in your program. The format is:

```
CLOSE(relation-name [,ERR=s])
```

where

s = label of an executable statement to which control transfers on close error

The following statement closes the realms joined in relation REL3, which is shown in sample sub-schema COMPARE (figure 3-16). If an error occurs on close, program execution continues at statement 60:

```
CLOSE(REL3,ERR=60)
```

If you include a CLOSE statement in your program for each realm in the relation, you do not need to include a relation CLOSE statement.

COMPARE

* SOURCE LISTING * (80351) DDLF 1.2+538.

```

00001      SUBSCHEMA COMPARE,SCHEMA=UNIVERSITY
00002
00003      ALIAS(REALM)  PFILE=PROFESSOR
00004      ALIAS(RECORD)  PRECORD=PROF-REC
00005      ALIAS(ITEM)    PROFID=PROF-ID.PROF-REC
00006      ALIAS(ITEM)    PNAME=PROF-NAME
00007
00008      ALIAS(REALM)    CRSFILE=COURSE
00009      ALIAS(RECORD)   CRSREC=COURSE-REC
00010      ALIAS(ITEM)    CRSID=COURSE-ID.COURSE-REC
00011      ALIAS(ITEM)    CRSNAME=COURSE-NAME
00012      ALIAS(ITEM)    PROF=PROF-ID.COURSE-REC
00013      ALIAS(ITEM)    FIELD=ACADEMIC-FIELD
00014
00015      ALIAS(REALM)    CFILE=CURRICULUM
00016      ALIAS(RECORD)  CRECORD=CURR-REC
00017      ALIAS(ITEM)    COURSE=COURSE-ID.CURR-REC
00018      ALIAS(ITEM)    CODE=COMPLETE-CODE
00019      ALIAS(ITEM)    DATE=COMPLETE-DATE
00020
00021      REALM PFILE
00022      REALM CRSFILE
00023      REALM CFILE
00024
00025      RECORD PRECORD
** WITHIN PFILE
00026      CHARACTER*8 PROFID
** ORDINAL 1
00027      CHARACTER*30 PNAME
** ORDINAL 2
00028      CHARACTER*20 FIELD
00029
** ORDINAL 3
00030      RECORD CRSREC
** WITHIN CRSFILE
00031      CHARACTER*6 CRSID
** ORDINAL 1
00032      CHARACTER*20 CRSNAME
** ORDINAL 2
00033      CHARACTER*8 PROF
00034
** ORDINAL 3
00035      RECORD CRECORD
** WITHIN CFILE
00036      CHARACTER*14 IDENT
** ORDINAL 1
00037      CHARACTER*6 COURSE
** ORDINAL 2
00038      CHARACTER*1 CODE
** ORDINAL 3
00039      CHARACTER*8 DATE
** ORDINAL 4
00040      REAL GRADE
00041
** ORDINAL 5
00042      RELATION REL3
PRIMARY KEY 00026      PROFID FOR AREA PFILE
ALTERNATE KEY 00028    FIELD FOR AREA PFILE
PRIMARY KEY 00031      CRSID FOR AREA CRSFILE
ALTERNATE KEY 00033    PROF FOR AREA CRSFILE
PRIMARY KEY 00036      IDENT FOR AREA CFILE
ALTERNATE KEY 00037    COURSE FOR AREA CFILE
ALTERNATE KEY 00040    GRADE FOR AREA CFILE
*****
*****      RECORD MAPPING IS NOT NEEDED FOR REALM - PFILE
*****      RECORD MAPPING IS NEEDED FOR REALM - CRSFILE
*****      RECORD MAPPING IS NEEDED FOR REALM - CFILE
00043      RESTRICT CRECORD (CODE .EQ. 'C')
00044      END

```

Figure 3-16. Sub-Schema COMPARE (Sheet 1 of 2)

```

00045
*****      END OF SUB-SCHEMA SOURCE INPUT

RELATION 001      *****      RELATION  STATISTICS      *****
                                REL3 JOINS      AREA - PFILE
                                                AREA - CRSFILE
                                                AREA - CFILE

                                Library name SSLIB
                                Privacy key 'XX99' needed for CFILE.

```

Figure 3-16. Sub-Schema COMPARE (Sheet 2 of 2)

READING A RELATION

To read a relation, you must include a relation READ statement in your program. The format is:

```
READ(relation-name [,KEY symbol item-name]
    [,ERR=s] [,END=s])
```

where

symbol = .EQ. .GT. .GE.

item-name = primary or alternate key

s = label of an executable statement to which control transfers on read error (ERR)

label of an executable statement to which control transfers on end-of-file (END)

When you omit the KEY option, the read operation is sequential. When you include the KEY option, the read operation is random.

The END option is valid only for a sequential read operation.

Sequential Relation Read

A sequential relation read accesses the relation occurrence located at the current relation position. Successive read operations return relation occurrences by position of the root realm, which is the first realm listed for the relation in the Relation Statistics portion of the sub-schema listing. Indexed sequential files are sequenced in ascending primary key order, actual key files are sequenced by block and record slot within the block, and direct access files are sequenced by position in home blocks.

A relation occurrence is composed of record occurrences. A tree structure of record occurrences for relation REL3 is shown in figure 3-17. Assuming that A1 is the first record in PFILE, the first and subsequent sequential reads return record

occurrences to the working storage area in the following order: A1B1C1, A1B1C2, A1B1C3, A1B2C4, and so forth. When record occurrences in CRSFILE and CFILE are exhausted, a subsequent sequential read returns the next record (A2, not shown) in PFILE and associated records in CRSFILE and CFILE as the operation repeats.

Typical FORTRAN 5 statements issued outside of a data base environment terminate with a fatal error if EOF is sensed and a test for EOF status is not included in the FORTRAN READ statement. If EOF status is not tested in a DML READ statement and EOF is sensed, program execution continues with the next statement. Consequently, it is necessary to test for EOF on a DML sequential read operation. This can be handled in one of two ways:

- Include the END option on the READ statement. When an EOF is sensed, program execution continues at the statement specified in the option.
- Include a test for an EOF value of 100g in the data base status block. This option is described in section 4.

In a program using sample sub-schema COMPARE (figure 3-16), a sequential read appears as shown in figure 3-18. If an error occurs on read, program execution continues at statement 600. The END option is included on the READ statement to test for EOF. When EOF is reached, program execution continues at statement 900.

The sequential read returns the first record in PFILE and the first corresponding record occurrences in CRSFILE and in CFILE. Successive reads return qualifying record occurrences as indicated in the preceding discussion of the tree structure of record occurrences. If EOF is sensed on PFILE, the relation read transfers control to the statement specified by the END option.

Notice the PRIVACY statement. Since CFILE is joined in the relation and has controlled access, the privacy key for that realm is required.

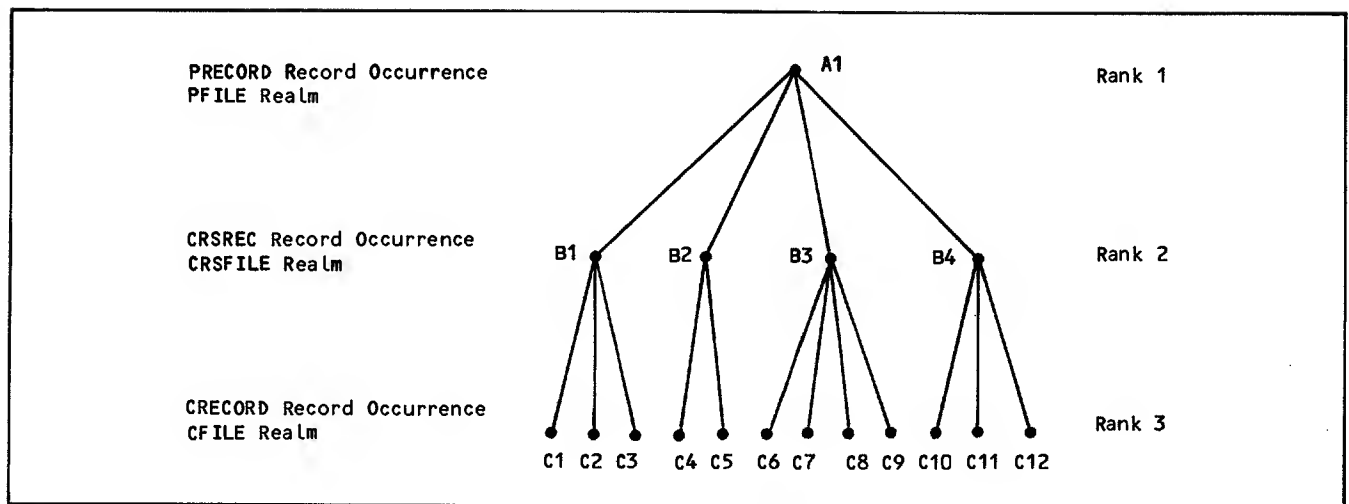


Figure 3-17. Tree Structure of Record Occurrences

```

PROGRAM RELMOD
SUBSCHEMA(COMPARE)
INVOKE
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(REL3,MODE=I,ERR=100)
READ(REL3,ERR=600,END=900)
.
.
.
900 CLOSE(REL3)
TERMINATE
END

```

Figure 3-18. Reading a Relation Sequentially

```

PROGRAM RELMOD
SUBSCHEMA(COMPARE)
INVOKE
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(REL3,MODE=I,ERR=100)
PROFID='RSS00860'
READ(REL3,KEY=PROFID,ERR=600)
.
.
.
900 CLOSE(REL3)
TERMINATE
END

```

Figure 3-19. Reading a Relation Randomly

Random Relation Read

A random relation read accesses a relation occurrence by the value of a referenced primary or alternate key. The referenced key must be in the root realm. For a program using the sample sub-schema COMPARE, the key named in the READ statement must be associated with PFILE (the root realm) rather than CRSFILE or CFILE. The program must set the value of the referenced key before the READ statement is executed.

In a program using sub-schema COMPARE (figure 3-16), a random read appears as shown in figure 3-19. The program sets the primary key PROFID of PFILE to RSS00860. The random read returns the record occurrence in PFILE that has PROFID equal to RSS00860 and the first corresponding record occurrences in CRSFILE and CFILE.

Control Break

A control break occurs when a new record occurrence is read for a parent realm in a relation. Control break status, however, is returned for the realm of the child. Therefore, if a realm in a relation has control break status after execution of a sequential read, the record occurrence read for this realm is a child record occurrence for a new parent record occurrence.

In the example shown in the tree structure of record occurrences (figure 3-17), control break occurs when A1 is first read (when A1B1C1 is returned). In this situation, control break status is returned for CRSFILE (rank 2) and for CFILE (rank 3). A control break occurs when B2 is first read (when A1B2C4 is returned), when B3 is first read (when A1B3C6 is returned), and so forth. In these situations, control break status is returned for CFILE, which is rank 3 of the relation.

The presence of a control break and the rank of the realm that is the lowest ranked realm with control break status can be determined by checking the data base status block. Status checking is described in section 4.

Null Occurrence

A null occurrence denotes that either no record occurrence qualifies for a read or that a record occurrence does not exist at a given level in a relation.

A read relation operation produces a null occurrence when one of the following is true:

- A parent record occurrence qualifies for the read, but no child record occurrence qualifies.
- A parent record occurrence qualifies for the read, but no child record occurrence exists.

If a null record occurrence is returned for each realm in a relation except the root realm, another READ statement must be executed to obtain the next set of record occurrences.

A null occurrence consists of a display code right bracket (J) in each character position of the record in the working storage area. The presence of a null occurrence and the lowest rank on which it occurred can be detected by checking the data base status block. Status checking is described in section 4.

Some examples of null record occurrences returned are shown in figure 3-20. In the first example, the lowest rank with a null record occurrence is rank 2. In the second and third examples, the lowest rank with a null record occurrence is rank 3.

POSITIONING A RELATION

To position a relation for subsequent sequential read operations, you must include a START statement in your program. The format is:

```
START(relation-name [,KEY symbol item-name]
      [,ERR=s])
```

where

symbol = = .EQ. .GT. .GE.

item-name = primary or alternate key defined in the root realm

s = Label of an executable statement to which control transfers on start error

Before the START statement is executed, the realm must have been opened for input or input/output.

When you omit the KEY option, the relation is positioned by primary key value of the root realm; the root realm is positioned to the record occurrence with a primary key value equal to the current value of the primary key item. When you include the KEY option, the root realm is positioned to the first record occurrence with a matching key value.

In a program using the sample sub-schema COMPARE (figure 3-16), both forms of the START statement appear as shown in figure 3-21. If an error occurs on start, program execution continues at statement 700.

The first START statement omits the KEY option, which means the relation occurrence will be positioned by the root realm primary key. The primary key (PROFID) of the root realm is set to the value MLN00840 before the START statement is executed; the relation is positioned to the root realm record occurrence with the matching primary key.

The second START statement includes the KEY option. Alternate key FIELD is set to the value PSYCHOLOGY. The first record occurrence in PFILE with an alternate key equal to PSYCHOLOGY is the one to which root realm PFILE is positioned. The subsequent sequential reads reference the alternate key FIELD. These reads return record occurrences in the root realm in alphabetical order (collating sequence order) according to the value of the alternate key FIELD.

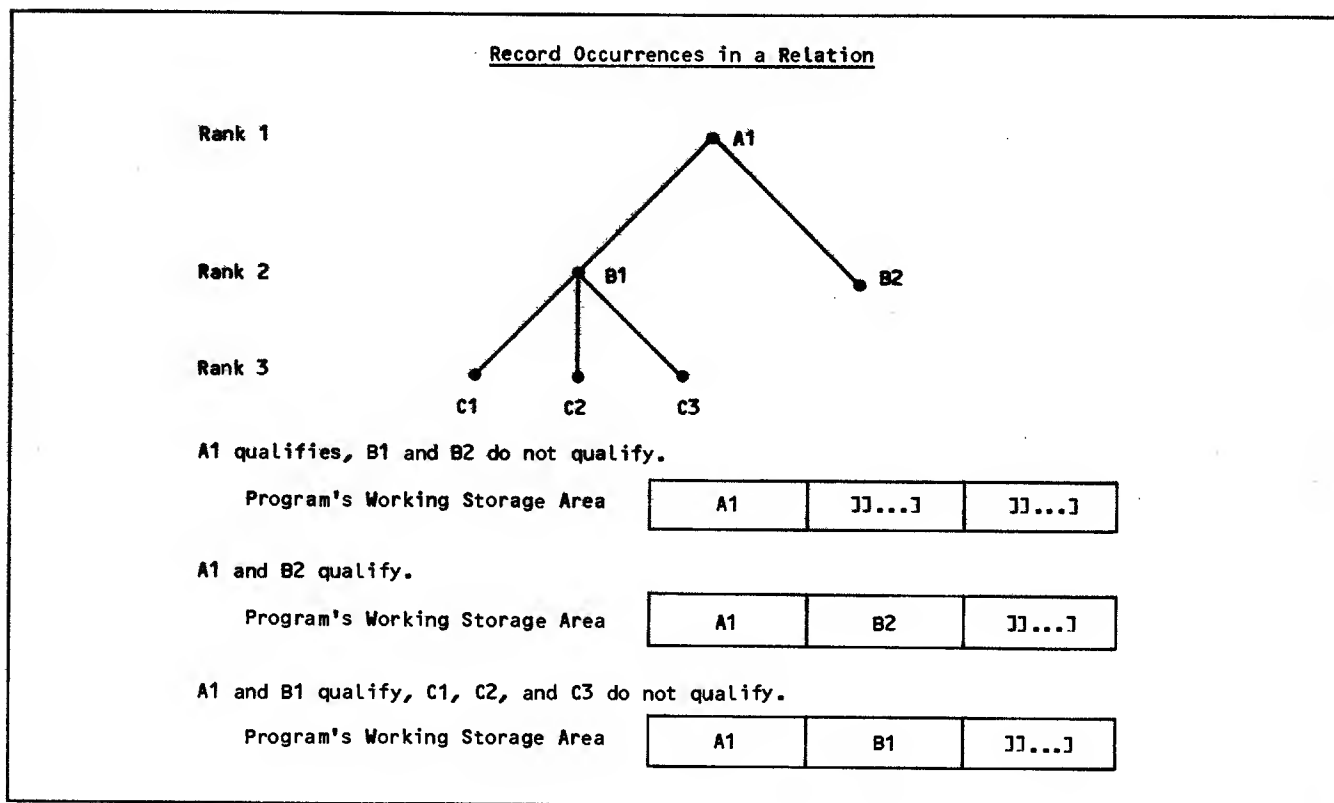


Figure 3-20. Null Record Occurrence Examples

```

PROGRAM RELMOD
SUBSCHEMA(COMPARE)
INVOKE
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(REL3,MODE=I,ERR=100)
PFOFID='MLN00840'
START(REL3,ERR=700)
READ(REL3,ERR=600,END=900)
.
.
.
FIELD='PSYCHOLOGY'
START(REL3,KEY.EQ.FIELD,ERR=700)
READ(REL3,ERR=650,END=750)
.
.
.
900 CLOSE(REL3)
TERMINATE
END

```

Figure 3-21. Positioning a Relation

UPDATING REALMS JOINED IN A RELATION

Realms joined in a relation can be updated, but care should be exercised. Related files are joined in the schema by a common data item to form a parent/child relationship. The schema contains a

JOIN clause in which a data item in one realm is equated with an identical data item in another realm. This common data item is called a join item.

CDCS normally does not monitor update operations that would alter the underlying relationship between related files. The exception is when constraints have been incorporated in the schema by the data administrator, as described in section 4.

Assuming constraints are not present, the following precautions should be noted:

- Modification of join item values can change parent/child relationships.
- Deletion of parent record occurrences can make all child record occurrences of the deleted parent record occurrence inaccessible when a relation is read.

Important rules to remember for relation update are:

- Always delete a child occurrence before deleting the parent record occurrence.
- Always write the parent record occurrence before writing a child record occurrence.
- Be aware of file positioning; input/output operations could alter positions on the files joined in the relation while within a sequential read loop.

O

O

O

O

O

O

O

ERROR PROCESSING AND STATUS HANDLING TECHNIQUES

4

DMS-170 offers a variety of error and status processing mechanisms. Each serves a specific purpose in the operating environment. These mechanisms are summarized in table 4-1 and detailed in the following paragraphs.

USING ERR AND END PROCESSING OPTIONS

A transfer of control to special processing for error or end-of-file (EOF) conditions can be specified in your program. This is accomplished by including the ERR and END options in the appropriate DML statements.

The ERR option can appear in the following statements:

OPEN	CLOSE	START
LOCK	UNLOCK	READ
REWRITE	WRITE	DELETE

The END option can appear in the following statement:

READ (sequential only)

The formats for the ERR and END options are:

ERR=statement-label

END=statement-label

When the ERR or END option is executed, control transfers to the statement identified by statement-label. The identified statement must be executable.

Assume an input/output error occurred during execution of the following statement:

OPEN(FILEX,ERR=50)

Execution of the OPEN statement is terminated, status is set to the appropriate error code as described later in this section, and program execution continues at statement 50.

Assume an EOF was sensed during execution of the following statement:

READ(FILEX,END=75)

Execution of the READ statement is terminated and execution continues at statement 75.

TABLE 4-1. ERROR AND STATUS PROCESSING MECHANISMS

Mechanism	Definition	Programmer Action
Error processing option	Syntax option that passes control to program logic on an error condition. Control is not passed when a CDCS relation condition indicating a null record occurrence or control break occurs.	Include ERR option in appropriate DML statements.
End-of-file processing option	Syntax option that passes control to program logic on an EOF condition.	Include END option in appropriate DML statements.
Status block	An array to which CDCS returns data base status information.	Include the following operations in the program: establishing the data base status block, calling subroutine DMLDBST once, and testing the status block contents at appropriate points.
Recovery points	Defined points to which the data base can be recovered with no loss of data.	Include a call to subroutine DMLRPT at appropriate points in the program.
Constraint handling	A method of avoiding situations in which constraints could be violated.	Be aware of constraints, and follow the rules for modifying the files on which constraints have been imposed.
Deadlock processing	A method of recovering from a situation in which programs are contending for locked resources.	If the program must have locks on several resources, include a test for deadlock status and provide program logic to re-establish any released locks.

Several examples of this type of error and end-of-file processing appear throughout section 3.

ESTABLISHING A DATA BASE STATUS BLOCK

An array called a data base status block can be established in your program to receive data base status information. When the status block is included in your program, CDCS updates the block after every operation on a realm or a relation.

The minimum length of the data base status block is one word; the maximum length is 11 words. You can include some or all of the words for testing purposes. The content of the status block is shown in table 4-2.

The following rules apply to the status block:

- Only one status block can exist at a time in the program.
- The status block must be declared as type **INTEGER**.
- The length of the block must be sufficient to completely include each desired portion of status information.

The following declaration would provide a complete status block:

```
INTEGER STATUS(11)
```

The following declaration would provide a 5-word status block reflecting all information except that pertaining to relation processing:

```
INTEGER STAT(5)
```

The location and length of the status block are conveyed to CDCS through a call to the **DMLDBST** routine. The routine can be called at any point in the program after the **INVOKE** statement. The format of the call is:

```
CALL DMLDBST(block-name,length)
```

where

block-name = name of the status block

length = length in words of the status block

The following rules apply to the **DMLDBST** routine:

- The routine needs to be called one time only.
- The call to **DMLDBST** should appear before the first DML statement after **INVOKE**. Positioning of the **DMLDBST** call is important because the call initializes the status block to zeros and blanks.
- If **DMLDBST** is called more than once in a program, the status block defined in the last call is the one that is updated by CDCS.

In a program using the sample sub-schema **COMPARE** shown in section 3, a data base status block declaration appears as shown in figure 4-1. The formats for printing the contents of the data base status block are also shown in the figure.

```
PROGRAM DBSEXP
INTEGER STATBLK(11)
SUBSCHEMA(COMPARE)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(REL3,MODE=I,ERR=100)
PFOFID='MLN00840'
READ(REL3,ERR=600,END=900)
.
.
.
600 PRINT *, 'ERROR ON READ'
PRINT 700, STATBLK
700 FORMAT (1X, 'STATUS BLOCK' /
1          1X,04,2X,I5,2X,03,2X,I2,2X,
1          A10,2X,I5,2X,I5,2X,I5,2X,A30)
900 CLOSE(REL3)
TERMINATE
END
```

Figure 4-1. Establishing a Data Base Status Block

ERROR CHECKING

Error checking should be performed after every operation on a realm or relation. Two methods are available:

- Test the error code in word 1 of the data base status block after every operation. For example:

```
OPEN(CFILE)
IF(STATBLK(1) .NE. 0)...
```

- Include the **ERR** option on the DML statement as appropriate and handle status block printing in one specific section of the program. For example:

```
OPEN(CFILE,ERR=50)
.
.
.
50 PRINT 60,STATBLK
```

STATUS CHECKING

Status checking should be performed as appropriate during relation processing to determine control breaks and null occurrences. Testing is performed on words 7 and 8, respectively, of the data base status block. (For more information about control break and null occurrence, see section 3.)

Word 7 indicates the lowest rank on which a control break occurred. A nonzero value in this word indicates a control break. To test for a control break, you can include a test on word 7 in your program. For example:

```
READ(CFILE)
IF(STATBLK(7) .NE. 0)...
```


TABLE 4-2. STATUS BLOCK CONTENT

Word	Content	Comments
1	The CDCS or CRM octal error code for the last data base operation on a realm or a relation.	Only error codes are returned. Status codes indicating null occurrences or control breaks are not returned in this word. A zero value indicates no error occurred. Use 0 format for printing.
2 [†]	A sub-schema item ordinal for CDCS errors occurring at the item level.	Item-level errors are associated with data validation and data base procedures established by the data administrator in the schema, and with CDCS record mapping. A zero value indicates no error occurred. Use I format for printing.
3 [†]	A CRM octal code indicating file position of the realm when the last data base operation was performed. The code is returned for open, close, read, and start operations. For a relation operation, the code indicates the file position of the root realm.	Code values are: 01g Beginning-of-information. 10g End-of-keylist. The last primary key value associated with a given alternate key was returned during a read operation using an alternate key value. 20g End-of-record. A record was returned during a read operation. 100g End-of-information. A sequential read operation was attempted after the previous operation returned the last record in the realm. Use 0 format for printing.
4 [†]	Not used (reserved for CDCS).	
5	The name of the function being performed when an error or relation condition occurred; the name is left-justified and blank filled.	If no error has occurred, this word contains no valid information. Use A10 format for printing.
6 ^{††}	The rank of the realm on which a CDCS or CRM error occurred during a relation operation. (The ranks of realms joined in a relation are numbered consecutively, with the root realm having rank 1.)	A zero value indicates no error occurred. Use I format for printing.
7 ^{††}	The lowest rank on which a control break occurred during a relation operation.	All realms in the relation with a rank greater than the rank stored in this word also have control breaks or null status. (Null status overrides control break status.) A zero value indicates no control break occurred. Use I format for printing.
8 ^{††}	The lowest rank for which there was a null record occurrence during a relation operation.	All realms in the relation with a rank greater than the rank stored in this word also have null occurrences. A zero value indicates no null occurrence. Use I format for printing.
9,10,11	The name of the realm on which an error occurred; the name is left-justified and blank filled.	A blank value indicates no error occurred; a blank value also can indicate the error occurred on an operation not associated with input/output or occurred on an input/output operation not explicitly requested by the application program. Use A30 format for printing.
[†] Words 2, 3, and 4 are treated as a single unit by CDCS; length must be provided for all three words if information for any portion of the unit is to be returned. ^{††} Words 6, 7, and 8 must all be defined to obtain any one word of relation status information.		

Word 8 indicates the lowest rank for which there was a null occurrence. A nonzero value in this word indicates a null occurrence. Since the right bracket character (]) is stored in a null record, you would probably want your program to bypass printing or move spaces to the print line. To test for a null occurrence, you can include a test on word 8 in your program. For example:

```
READ(CFILE)
IF(STATBLK(8) .NE. 0)...
```

DEFINING RECOVERY POINTS

Protection of the data base is a primary concern of the data administrator. This individual employs all of the available CDCS resources to guarantee the data base is not inadvertently lost or destroyed as a result of system failure or entry of incorrect data. Although you might not be aware of it, record images are being recorded on special files called log files during data base processing. If problems develop, these log files are used by the data administrator to recreate all or specified portions of a data base.

Sometimes it is important to select certain points in an application program to which the data base can be recovered and the application program successfully restarted. Incorporating these recovery points incurs a considerable amount of overhead and if used extensively, can impact throughput; therefore, the determination to include recovery points is always made by the data administrator.

Recovery points can be established in your program through a call to the DMLRPT routine. The call marks the point for recovery purposes. The format of the call is:

```
CALL DMLRPT(number,comment)
```

where

number = the name of an integer field to which CDCS is to assign a unique recovery point number

comment = the name of an array containing a 30-character explanatory comment about the recovery point

It is your responsibility to establish the number field and to declare the field as type INTEGER. You might want your program to save this number for reference purposes. It is also your responsibility to establish the comment field and to declare the field as a 3-word array.

In a program using the sample sub-schema AVERAGE shown in section 3, a recovery point definition appears as shown in figure 4-2. Routine DMLRPT is called immediately after a write operation. The number that CDCS assigned is printed out in the example only for reference purposes. Execution of DMLRPT causes the application program to be suspended and the following events to occur in the order given:

1. All input/output buffers for schema data base areas associated with sub-schema AVERAGE are flushed.
2. A recovery point log record is written to the log file for the data base. The CDCS-assigned recovery point number (NUM) and the comment PROGRAM RCVEXMP are stored in the log record.

```
PROGRAM RCVEXMP
INTEGER STATBLK(11)
INTEGER NUM(10)
CHARACTER *30 CMT
SUBSCHEMA(AVERAGE)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
CMT='PROGRAM RCVEXMP'
IDENT='122-13-6704-10'
STUDENT='122-13-6704'
COURSE='CHM103'
WRITE(CFILE,ERR=500)
CALL DMLRPT(NUM,CMT)
30 PRINT 40,NUM
40 FORMAT(1X,I10)
.
.
.
900 CLOSE(CFILE)
TERMINATE
END
```

Figure 4-2. Defining Recovery Points

After execution of the DMLRPT routine, you are assured the data base can be recovered to its current state.

AVOIDING CONSTRAINT VIOLATIONS

The data administrator incorporates constraints in the schema for the purpose of protecting interdependent data. Constraints can be defined for two logically associated items within a single file (single-file constraint) and for two logically associated items within two files (two-file constraint).

Consider an employment file in which each record occurrence contains an employee number and a manager number, where the manager number conforms to the structure of the employee number. Figure 4-3 illustrates this concept. Assume, for example, the data administrator designed the schema with the following single-file constraint:

MNGR-NO DEPENDS ON EMP-NO

In this example, MNGR-NO (the child item) is dependent upon EMP-NO (the parent item). This means that no occurrence of the child record can exist in the data base unless an occurrence of the parent record also exists with the same value of the associated data item. Also, no parent record can be deleted if a child record exists with the same value of the associated data item.

The parent item in a single-file constraint is always a primary key or an alternate key with no duplicates; the child item is a primary key or an alternate key, and the alternate key can have duplicates. You would violate the constraint presented in the example if you attempted to do any of the following:

- Store an employee EMPLOYMENT record if an EMPLOYMENT record for the referenced manager does not exist. (An organization could not recognize a manager who was not first an employee.)
- Change the value of the parent item (EMP-NO) if a corresponding child item (MNGR-NO) exists. (An organization could not change an employee number as long as references to the old number existed.)
- Delete a manager EMPLOYMENT record if an employee EMPLOYMENT record with the corresponding manager number exists. (An organization could not remove a manager while an employee was still reporting to that individual.)

In a single-file constraint, at least one record exists that has no parent record. This situation occurs in the single-file constraint example for the employee who has no manager. The record for this employee must have the same value for both EMP-NO and MNGR-NO.

If you are creating a file on which a single-file constraint has been imposed, take the following steps in the order given:

1. Create the file with record occurrences of the items that have no parent.
2. Close the file.
3. Reopen the file for input/output and add the record occurrences of the child items. (Ensure that a parent record occurrence exists before adding any corresponding child item.)

For a situation involving a two-file constraint, consider a course file and a curriculum file. Assume that the data administrator designed the schema with the following two-file constraint:

COURSE-ID OF CURR-REC DEPENDS ON
COURSE-ID OF COURSE-REC

The records of the two files and the data items associated in the constraint are shown in figure 4-4. CURR-REC (the child record) is dependent upon COURSE-REC (the parent record) if there is a correspondence between them. A correspondence exists if the child record and the parent record each contain the same value for the common item, which is COURSE-ID in this example.

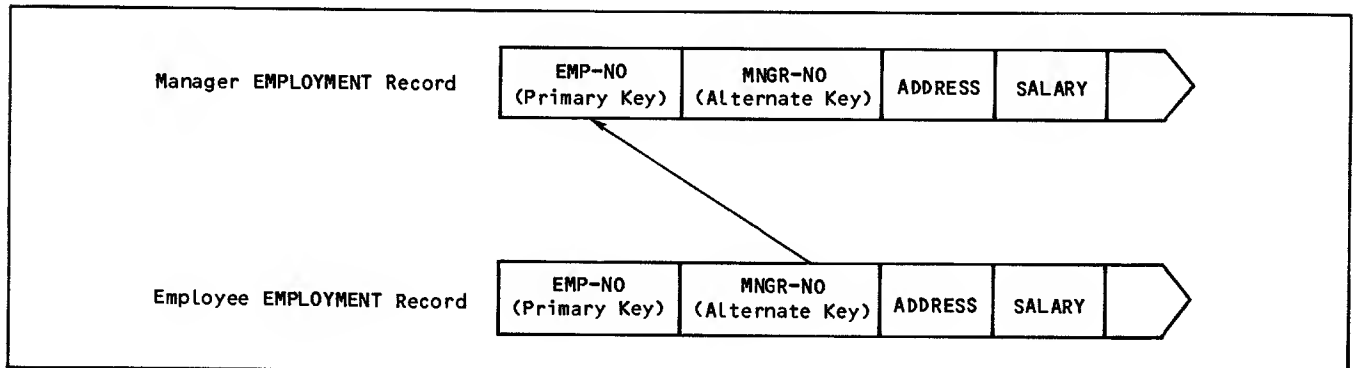


Figure 4-3. Single-File Constraint Example

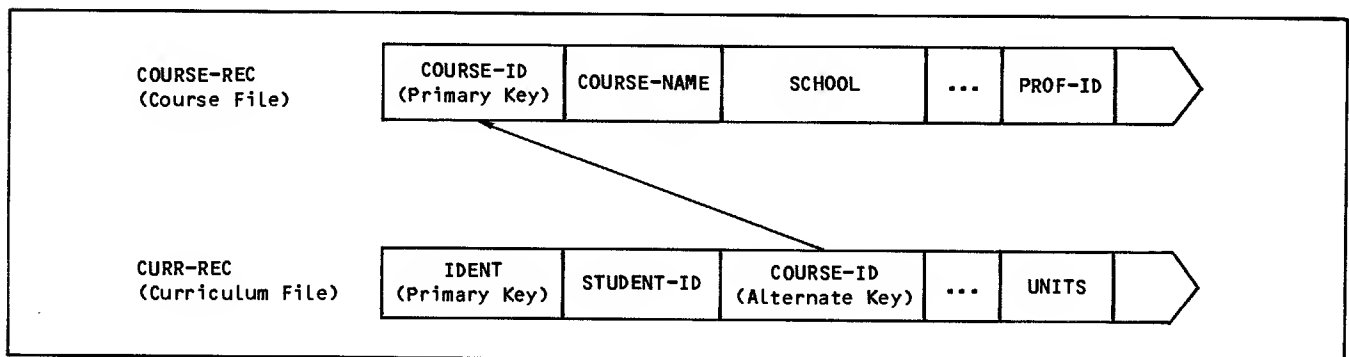


Figure 4-4. Two-File Constraint Example

You would violate the constraint presented in the two-file constraint example if you attempted to do any of the following:

- Delete a **COURSE-REC** occurrence if a corresponding **CURR-REC** occurrence exists. (The university could not drop a course from its curriculum while a student was still enrolled.)
- Change the **COURSE-ID** value of a **COURSE-REC** occurrence if a corresponding **CURR-REC** occurrence exists. (The university could not change the identification code of a course as long as a student's record still uses that code.)
- Add a **CURR-REC** occurrence if a corresponding **COURSE-REC** occurrence does not exist. (A student could not be enrolled in a course that was not being offered by the university.)

If you are modifying the common item of a file on which a two-file constraint has been imposed and the common item is a primary key, take the following steps in the order given:

1. Write the parent record with the new value in the common item.
2. Read a dependent child record, and change the value of the common item to the new value of the parent record. Rewrite the child record. (Perform this step for each child record of the parent.)
3. Delete the parent record with the old value.

If you are modifying the common item of a file on which a two-file constraint has been imposed and the common item is an alternate key, take the following steps in the order given:

1. Write each child record containing the old value of the item to a temporary file.
2. Delete each child record containing the old value of the item from the data base.
3. Read the parent record, and change the value of the data item to the new value. Rewrite the parent record.
4. Read a dependent child record from the temporary file, and change the value of the common item to the new value of the parent record. Write the child record to the data base. (Perform this step for each child record of the parent.)

Since constraints are established in the schema and not indicated in any way in the sub-schema, it is the responsibility of the data administrator to supply you with this information. By being aware of constraints, you can anticipate violations and prevent them from occurring in your application program.

When a constraint is violated, CDCS aborts the particular operation, returns a nonfatal 601g error code, and continues processing. The error message identifies the record on which the

attempted violation occurred. Whenever you are writing, deleting, or rewriting a record, the appropriate data base status block entry should be tested.

Two general rules to remember for constraint processing are:

- Always delete a child record occurrence before deleting the parent record occurrence.
- Always write the parent record occurrence before writing a child record occurrence.

ANTICIPATING DEADLOCK SITUATIONS

CDCS allows concurrent access to a data base. This means that two or more application programs can access the same file (realm) at the same time. The following can take place:

- Two or more application programs can open the same file for input and perform simultaneous read operations.
- One application program can open a file for input/output and perform update operations, while other programs can open the same file for input and perform simultaneous read operations.
- Two or more application programs can open the same file for input/output, but only one program can gain immediate access to a particular record to perform update operations.

The integrity of the data base is maintained through CDCS locking mechanisms: the record locking mechanism and the file locking mechanism. CDCS holds a lock for an application program and prevents update of the locked file or record by any other program.

An application program causes the CDCS record locking mechanism to be in effect by opening a file for input/output and not locking the file. When the CDCS record locking mechanism is in effect, CDCS always locks a record for an application program when that program reads a record. Other application programs can read the locked record but can not update the record. The record lock does not affect access to other records in the file. CDCS releases the record lock when the application program for which the lock is held rewrites or deletes the record, reads another record, or closes the file.

An application program causes the CDCS file locking mechanism to be in effect by issuing an explicit lock request (that is, by execution of the DML LOCK statement). When the CDCS file locking mechanism is in effect, CDCS locks the entire file for the requesting program. Other programs can read that file if they have opened it for input; programs that have opened that file for input/output can neither read nor update.

Table 4-3 lists the various locking operations and their effects.

TABLE 4-3. LOCKING OPERATIONS

Operation	Effect
An application program opens a realm for input/output and includes a DML LOCK statement. (This should be avoided whenever possible.	CDCS locks the entire realm against update by other users. An unlock or close operation by that program releases the lock.
An application program opens a realm for input/output without including a DML LOCK statement.	CDCS locks the record on the read operation. A rewrite, delete, or another read operation by that program releases the lock.
An application program opens a realm for output without including a DML LOCK statement.	CDCS locks the entire realm. A close operation by that program releases the lock.
An application program opens a relation for input/output.	CDCS locks all records in a given relation occurrence. A rewrite or delete operation by the program releases the lock on the record updated. The next relation read operation by that program releases the record locks on the files for which a new record has been read.

Whenever two or more application programs contend for locked resources, which are files or records, a deadlock situation can occur. Contention occurs when two programs, each having at least one resource locked, attempt to lock a resource that is locked by the other program. Neither program can continue processing, because neither program can obtain the necessary locks. CDCS automatically releases the locked resources of one program. The other program then can obtain the locks it requires and can continue processing.

When CDCS has detected a deadlock situation and has released the locked resources of an application program, CDCS issues the deadlock error status code 6638 to that program. If the application program established the data base status block, the program can check the first word for the deadlock code.

If your program must have locks on several resources, your program should always test for deadlock status before attempting to update a file. If deadlock occurs, your program should reestablish the locks that it held before continuing further processing.

An example illustrating deadlock processing appears in figure 4-5. Files joined in relation REL3 are

```

PROGRAM DEADLCK
INTEGER STATBLK(11)
SUBSCHEMA(COMPARE)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')
OPEN(REL3,ERR=100)
PROFID='JMS00160'
30 READ(REL3,KEY=PROFID)
IF(STATBLK(1) .EQ. 0"663") GO TO 30
.
.
.
900 CLOSE(REL3)
TERMINATE
END

```

Figure 4-5. Deadlock Processing

opened for input/output. The program presumably is reading a record prior to update and CDCS has locked all records in the relation occurrence. The example includes a test of word 1 in the status block to enter a loop in case of deadlock. In the loop, the program attempts to reestablish the locks and checks for deadlock.

0

0

0

0

0

0

0

FORTRAN application programming in the DMS-170 environment relieves you of several responsibilities. For example:

- You do not have to describe data within your program; the data administrator incorporates data descriptions in the schema and sub-schema. Data descriptions in a sub-schema are included in your program.
- You do not have to write conversion routines; CDCS handles all conversion for you.
- You do not have to write all routines that perform validity checking; the data administrator generates data base procedures, which are specified in the schema and called at appropriate times.
- You do not have to write separate logging and recovery utilities; the data administrator provides for data base restoration by specifying logging operations in the master directory.
- You do not have to be concerned with the details of input/output; CDCS handles them.

DEVELOPING AN APPLICATION PROGRAM

To develop a DMS-170 application program, you must do the following:

- Obtain a listing of the sub-schema from the data administrator.
- Obtain the name of the sub-schema library from the data administrator.
- Obtain the appropriate privacy keys from the data administrator.
- Be aware of any constraints that have been incorporated in the schema.
- Include appropriate DML statements in your FORTRAN program.

COMPILING AND EXECUTING THE SOURCE PROGRAM

To compile and execute a DMS-170 FORTRAN application program, you must do the following:

1. Attach the sub-schema library for DML preprocessing of the source program.
2. Include a DML control statement for DML preprocessing of the source program.
3. Include an FTN5 control statement that specifies the DML output file as the input file for the FORTRAN 5 compiler.
4. Include an LDSET control statement for loading the system library for execution of the source program.
5. Include the name of the file containing the relocatable binary program (LG0 is the default name) to execute the program.

DML statements are preprocessed before source program compilation. The DML preprocessor translates the DML statements into appropriate FORTRAN statements. When translation is complete, the DML preprocessor writes the FORTRAN source program to an output file with the default name DMLOUT. This output file, complete with translated DML statements, becomes the input file to the FORTRAN compiler. A block diagram illustrating FORTRAN/DML preprocessing is shown in figure 5-1.

The DML control statement calls the DML preprocessor. A list of DML control statement parameters is shown in figure 5-2.

The statements required to execute the DML preprocessor and to compile the source program are shown in figure 5-3.

The statements required to execute the DML preprocessor and to compile and execute the source program are shown in figure 5-4. An LDSET control statement naming the system library, DMSLIB, must be included for program execution.

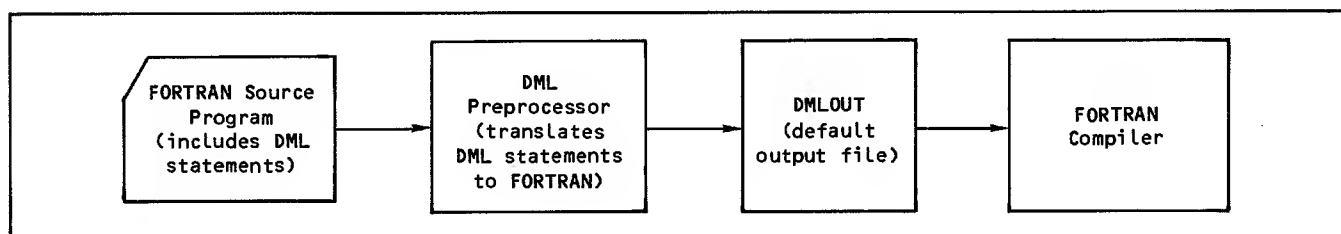


Figure 5-1. FORTRAN DML Preprocessing

DML(p1,p2,p3,p4,p5,p6,p7)		
p1	SB=lfm	Name of file containing sub-schema library.
	SB	Same as SB=SBLFN.
	SB=0	Not allowed.
	omitted	Same as SB=SBLFN.
p2	LV=F5	Specifies FORTRAN 5.
	LV	Same as LV=F5.
	omitted	Dependent on installation.
p3	I=lfm	Name of file containing FORTRAN source program with added DML statements to be preprocessed by DML.
	I	Same as I=COMPILE.
	omitted	Same as I=INPUT.
	I=0	Not allowed.
p4	O=lfm	Name of file to which translated version of FORTRAN source program is to be written. DML statements appearing in FORTRAN program are translated into FORTRAN statements before being written to this file.
	O	Same as O=DMLOUT.
	omitted	Same as O=DMLOUT.
	O=0	No output is produced.
p5	E=lfm	Name of file to which error diagnostics are to be written.
	E	Same as E=ERRS.
	omitted	Same as E=OUTPUT.
p6	ET=op	Error termination code. Four levels of errors are defined; if an error of the specified level or higher takes place, the job is aborted to an EXIT(S) control statement (NOS/BE) or EXIT control statement (NOS). The abort does not take place until DML is finished. The possible values for op, in increasing order of severity, are as follows:
		<p>T Trivial. The syntax of the usage is correct, but it is questionable.</p> <p>W Warning. The syntax is incorrect, but the processor has been able to recover by making an assumption about what was intended.</p> <p>F Fatal. An error prevents DML from processing the statement in which it occurs. Unresolvable semantic errors also fall into this category. Processing continues with the next statement.</p> <p>C Catastrophic. Compilation cannot continue; however, DML advances to the end of the current program unit and attempts to process the next program unit.</p> <p>ET Same as ET=F.</p> <p>omitted Same as ET=0.</p> <p>ET=0 The job step is not to be aborted even if errors occur (except for control statement errors).</p> <p>T and W errors do not invalidate the output file produced by DML (the file specified by the O option). The translated code on the file can still be compiled (barring any errors not related to DML), but the results might not be what the user intended. F and C errors, however, produce an output file that cannot be successfully compiled by FORTRAN.</p>
p7	DS	Directive suppression. Listing control directives are not generated; all FORTRAN statements generated by DML preprocessing appear in the FORTRAN source listing.
	omitted	Listing control directives are generated; FORTRAN statements generated by DML preprocessing of the SUBSCHEMA and INVOKE statements do not appear in the FORTRAN source listing.
		FORTRAN CALL statements generated as a result of executable DML statements always appear on the FORTRAN source listing regardless of DS specification.

Figure 5-2. DML Control Statement


```

Job statement
USER statement }
CHARGE statement } NOS only
ACCOUNT statement } NOS/BE only
ATTACH(sub-schema-library)
DML(SB=sub-schema-library,LV=F5)
FTN5(I=DMLOUT)
End-of-record
FORTRAN source program containing DML statements
End-of-information

```

Figure 5-3. Executing DML and Compiling the Source Program

```

Job statement
USER statement }
CHARGE statement } NOS only
ACCOUNT statement } NOS/BE only
ATTACH(sub-schema-library)
DML(SB=sub-schema-library,LV=F5)
FTN5(I=DMLOUT)
LDSET(LIB=DMSLIB)
LGO.
End-of-record
FORTRAN source program containing DML statements
End-of-information

```

Figure 5-4. Compiling and Executing the Source Program

SAMPLE PROGRAMS

Sample programs appear in the remainder of this section. Each program uses the data base environment that is established and illustrated in appendix C. You should read this appendix to become familiar with the schema, sub-schemas, and stored data before examining the FORTRAN programs.

When the DML preprocessor translates DML statements into FORTRAN statements, the FORTRAN statements can be printed out or suppressed, depending on the setting of the DS parameter on the DML control statement. When the DS parameter is included, all FORTRAN statements generated by the DML preprocessor appear in the FORTRAN source listing. When the DS parameter is omitted, listing control statements are generated and inserted immediately after the SUBSCHEMA and INVOKE statements; therefore, the FORTRAN statements generated by DML preprocessing of these statements do not appear in the FORTRAN source listing.

Listing control directives appear in the sample program source listings in the following form:

```

C$ LIST(ALL=0)
C$ LIST(ALL)

```

These directives are generated automatically by the DML preprocessor. They appear because the DS parameter in each DML control statement was omitted. Notice, however, that CALL statements generated as a result of executable DML statements appear regardless of the DS parameter setting.

Each sample program is illustrated by including the control statements, the source program statements, the compilation listing, and the output of program execution. The programs are:

Program RATING	Figure 5-5
Program INDAVGE	Figure 5-6
Program RELATE	Figure 5-7
Program CHARGES	Figure 5-8
Program ADMIT	Figure 5-9

Control Statements

Job Statement
USER statement
CHARGE statement
ATTACH(SSLIB)
DML(SB=SSLIB,LV=FS)
FTN5(I=DMLOUT)
LDSET(LIB=DMSLIB)
LGO.
End-of-record

Source Program

PROGRAM RATING

C THIS PROGRAM READS ALL STUDENT GRADES AND CALCULATES THE
C AVERAGE FOR THE SCHOOL. THE PROGRAM PERFORMS A SEQUENTIAL
C READ OF ALL STUDENT GRADES. THE END PARAMETER ON THE DML
C READ STATEMENT TRANSFERS CONTROL TO STATEMENT 70 ON EOF.
C THE PROGRAM THEN CALCULATES THE AVERAGE AND PRINTS OUT
C THE SOLUTION.
C

INTEGER STATBLK(11)
SUBSCHEMA(AVERAGE)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')
OPEN(CFILE,ERR=50)
N=0

TOTAL=0
20 READ(CFILE,ERR=50,END=70)
IF(GRADE.EQ.0.0) GO TO 20
N=N+1

TOTAL=TOTAL + GRADE
GO TO 20

50 PRINT 60,STATBLK

60 FORMAT (1X, 'STATUS BLOCK' /
1 1X,04,2X,15,2X,03,2X,12,2X,
1 A10,2X,15,2X,15,2X,15,2X,A30)
GO TO 80

70 TOTAL=TOTAL / N

PRINT 75,TOTAL

75 FORMAT (1X, 'AVERAGE IS' /
1 1X,F4.1)

80 CLOSE(CFILE)

TERMINATE

END

End-of-record

Figure 5-5. Program RATING (Sheet 1 of 3)

483500 A

PROGRAM RATING 76/176 OPT=0

5-5

Figure 5-5. Program Rating (Sheet 2 of 3)

```

--VARIABLE MAP--(LO=A)
--NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
  COURSE      2B /DB0001/
  DBF0001     15B /DB0000/
  DBI0001     0B /DB0001/  EQV
  DBREALM     0B /DB0000/
  DBRELST     10B /DB0000/
  DBRUID      7B /DB0000/
  IDENT       0B /DB0001/  EQV
  N           220B
  STATBLK    205B

  DBR0001     11B /DB0000/
  DBSCNAM     4B /DB0000/
  DBSTAT      3B /DB0000/
  DBS0001     14B /DB0000/
  DBT0001     60B /DB0000/
  GRADE       0B /D0001AA/
  STUDENT     1B /DB0001/
  TOTAL       221B

  CHAR*6      35
  CHAR*1      3
  INTEGER     3
  INTEGER     1
  INTEGER     11
  INTEGER     11

--PROCEDURES--(LO=A)
--NAME--TYPE--ARGS--CLASS--
  DMLCLS      2 SUBROUTINE
  DMLDBST     2 SUBROUTINE
  DMLEND      0 SUBROUTINE
  DMLINV      6 SUBROUTINE

  DMLOPN      4 SUBROUTINE
  DMLPRV      8 SUBROUTINE
  DMLRD       6 SUBROUTINE
  LOCF        1 INTRINSIC

--STATEMENT LABELS--(LO=A)
--LABEL--ADDRESS--PROPERTIES--DEF
  20 40B
  50 56B
  60 110B  FORMAT

  73
  78
  79

  70 61B
  75 121B  FORMAT
  80 66B

--ENTRY POINTS--(LO=A)
--NAME--ADDRESS--ARGS--
  RATING      5B 0

--STATISTICS--
  PROGRAM-UNIT LENGTH      222B = 146
  SCM LABELLED COMMON LENGTH 67B = 55
  SCM STORAGE USED        60700B = 25024
  COMPILE TIME            0.039 SECONDS

Output From Program Compilation
AVERAGE IS
3.6

```

Figure 5-5. Program RATING (Sheet 3 of 3)

Control Statements

Job statement
USER statement
CHARGE statement
ATTACH(SSLIB)
DML(SB=SSLIB,LV=F5)
FTN5(L=DMLOUT)
LDSET(LIB=DMSLIB)
LGO.
End-of-record

Source Program

PROGRAM INDAVGE

C THIS PROGRAM READS ALL STUDENT GRADES AND CALCULATES
C INDIVIDUAL AVERAGES. THE FIRST READ SAVES THE FIRST
C STUDENT ID. WHEN THE STUDENT ID CHANGES, THE PROGRAM
C CALCULATES AND PRINTS THE AVERAGE, THEN CONTINUES TO
C READ THE FILE UNTIL EOF IS SENSED. TESTING THE THIRD
C WORD OF STATBLK CHECKS FOR EOF.

C INTEGER STATBLK(11)
C CHARACTER FINAL*3
C CHARACTER OLDID*11
C SUBSCHEMA(AVERAGE)
C FINAL='NO'

INVOKE

CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')

OPEN(CFILE,ERR=50)
READ(CFILE,ERR=50)

IF(STATBLK(3) .EQ. 0"100") GO TO 40
N=0

TOTAL=0.0

OLDID=STUDENT

20 IF(GRADE .GE. 1.0) THEN
N=N+1

TOTAL=TOTAL+GRADE

ENDIF

READ(CFILE,ERR=50)

IF(STATBLK(3) .EQ. 0"100") THEN
FINAL='YES'

ELSE

IF(STUDENT .EQ. OLDID) GO TO 20

ENDIF

IF(N .EQ. 0) GO TO 30

TOTAL=TOTAL / N

PRINT 25, OLDID,TOTAL

FORMAT ('0",9X,A11,3X,F4.1)

30 IF(FINAL .EQ. 'NO') GO TO 10

GO TO 100

40 PRINT *, 'EMPTY INPUT FILE'

50 PRINT 60,STATBLK

60 FORMAT (1X,'STATUS BLOCK' /

1 1X,04,2X,15,2X,03,2X,12,2X,

1 A10,2X,15,2X,15,2X,15,2X,A30)

100 CLOSE(CFILE)

TERMINATE

END

End-of-record

Figure 5-6. Program INDAVGE (Sheet 1 of 4)

Compilation Listing

PROGRAM INDAVGE 76/176 OPT=0 FTN 5.1+538

```

1  PROGRAM INDAVGE
2
3  C THIS PROGRAM READS ALL STUDENT GRADES AND CALCULATES
4  C INDIVIDUAL AVERAGES. THE FIRST READ SAVES THE FIRST
5  C STUDENT ID. WHEN THE STUDENT ID CHANGES, THE PROGRAM
6  C CALCULATES AND PRINTS THE AVERAGE, THEN CONTINUES TO
7  C READ THE FILE UNTIL EOF IS SENSED. TESTING THE THIRD
8  C WORD OF STATBLK CHECKS FOR EOF.
9  C
10 INTEGER STATBLK(11)
11 CHARACTER FINAL*3
12 CHARACTER OLDID*11
13 SUBSCHEMA(AVERAGE)
14 C$ LIST(ALL=0)
15 C$ LIST(ALL)
16 C$ FINAL='NO'
17 ** INVOKE
18 C$ LIST(ALL=0)
19 C$ LIST(ALL)
20 CALL DMLINV(0001,DBF0001,10HAVERAGE ,10H
+10H ,0"35514310376143061021")
21 CALL DMLDBST(STATBLK,11)
22 PRIVACY(CFILE,PRIVACY='XX99')
23 CALL DMLPRV(1,0,0001,
+0"60","XX99","" ,")
24 ** OPEN(CFILE,ERR=50)
25 CALL DMLOPN(DBF0001,0001,2HI0,*50 )
26 ** READ(CFILE,ERR=50)
27 CALL DMLRD(DBF0001,0001,1,0,*50 )
28 IF(STATBLK(3) .EQ. 0"100") GO TO 40
29 N=0
30 TOTAL=0.0
31 OLDID=STUDENT
32 IF(GRADE .GE. 1.0) THEN
33   N=N+1
34   TOTAL=TOTAL+GRADE
35 ENDIF
36 ** READ(CFILE,ERR=50)
37 CALL DMLRD(DBF0001,0001,1,0,*50 )
38 IF(STATBLK(3) .EQ. 0"100") THEN
39   FINAL='YES'
40 ELSE
41   IF(STUDENT .EQ. OLDID) GO TO 20
42 ENDIF
43 IF(N .EQ. 0) GO TO 30
44 TOTAL=TOTAL / N
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91

```

Figure 5-6. Program INDAVGE (Sheet 2 of 4)

```

92      PRINT 25, OLDID,TOTAL
93      FORMAT ("0",9X,A11,3X,F4.1)
94      IF(FINAL.EQ. 'NO') GO TO 10
95      GO TO 100
96      PRINT *, 'EMPTY INPUT FILE'
97      PRINT 60,STATBLK
98      FORMAT (1X,'STATUS BLOCK' /
99      1      1X,04,2X,15,2X,03,2X,12,2X,
100     1      A10,2X,15,2X,15,2X,15,2X,A30)
101     ** CLOSE(CFILE)
102     100 CALL DMLCLS(DBF0001,0001)
103     ** TERMINATE
104     CALL DMLEND
105     END

```

---VARIABLE MAP---(LO=A)

NAME	ADDRESS	BLOCK	PROPERTIES	TYPE	SIZE	NAME	ADDRESS	BLOCK	PROPERTIES	TYPE	SIZE
COURSE	28	/DB0001/		CHAR*6		DBT0001	60B	/DB0000/		INTEGER	2
DBF0001	158	/DB0000/		INTEGER	35	FINAL	314B			CHAR*3	
DBI0001	08	/DB0001/	EQV	CHAR*1		GRADE	0B	/D0001AA/		REAL	
DBREALM	08	/DB0000/		INTEGER	3	IDENT	0B	/DB0001/	EQV	CHAR*14	
DBRELST	10B	/DB0000/		INTEGER	1	N	317B			INTEGER	
DBRUID	7B	/DB0000/		INTEGER	3	OLDID	315B			CHAR*11	
DBR0001	11B	/DB0000/		INTEGER	3	STATBLK	301B			INTEGER	11
DBSCNAM	4B	/DB0000/		INTEGER	3	STUDENT	1B	/DB0001/		CHAR*11	
DBSTAT	3B	/DB0000/		INTEGER		TOTAL	320B			REAL	
DBS0001	14B	/DB0000/		INTEGER							

---PROCEDURES---(LO=A)

NAME	TYPE	ARGS	CLASS	NAME	TYPE	ARGS	CLASS
DMLCLS		2	SUBROUTINE	DMLPN		4	SUBROUTINE
DMLDBST		2	SUBROUTINE	DMLPRV		8	SUBROUTINE
DMLEND		0	SUBROUTINE	DMLRD		5	SUBROUTINE
DMLINV		6	SUBROUTINE	LOC	GENERIC	1	INTRINSIC

---STATEMENT LABELS---(LO=A)

LABEL	ADDRESS	PROPERTIES	DEF	LABEL	ADDRESS	PROPERTIES	DEF
10	50B		76	40	120B		96
20	55B		79	50	122B		97
25	154B	FORMAT	93	60	157B	FORMAT	98
30	114B		94	100	124B		102

Figure 5-6. Program INDAVGE (Sheet 3 of 4)

---ENTRY POINTS---(LO=A)
-NAME---ADDRESS---ARGS---

INDAVGE 5B 0

---STATISTICS---

PROGRAM-UNIT LENGTH 321B = 209
SCM LABELLED COMMON LENGTH 67B = 55
SCM STORAGE USED 60700B = 25024
COMPILE TIME 0.052 SECONDS

Output From Program Execution

100-22-5860	3.5
120-44-3760	3.1
122-13-6704	4.0
124-33-5780	3.8
197-11-2140	3.6
437-56-8943	3.3
120-44-3760	3.5
553-89-2021	3.8
678-12-1144	4.0
687-14-2100	3.9

Figure 5-6. Program INDVGE (Sheet 4 of 4)


```
Job statement
USER statement
CHARGE statement
ATTACH(SSLIB)
DML(SB=SSLIB,LV=F5)
FTN5(I=DMLOUT)
LDSET(LIB=DMSLIB)
LGO.
End-of-record
```

```

C      PROGRAM RELATE
C
C      THIS PROGRAM READS BY RELATION AND PRINTS GRADES FOR
C      STUDENT ID 120-44-3760.  THE FIRST READ IS RANDOM
C      FOLLOWED BY A TEST FOR NULL RECORD.  THE SECOND READ IS
C      SEQUENTIAL FOLLOWED BY A TEST FOR CONTROL BREAK.
C
      INTEGER STATBLK(11)
      CHARACTER SAVEKEY*11
      SUBSCHEMA(RELATION)
      INVOKE
      CALL DMLDBST(STATBLK,11)
      PRIVACY(CFILE,PRIVACY='XX99')
      OPEN(REL1,ERR=50)
      STID='120-44-3760'
      READ(REL1,KEY=STID,ERR=50)
      SAVEKEY=STID
      IF(SAVEKEY.EQ.'JJJJJJJJJJ') GO TO 70
      PRINT 20, CSTID,GRADE
      FORMAT ('100,9X,A11,3X,F4.1)
      READ(REL1,ERR=50,END=70)
      IF (STATBLK(7).NE.0) GO TO 70
      GO TO 15
      PRINT 60,STATBLK
      FORMAT ('1X,'STATUS BLOCK' /
      1      1X,04,2X,15,2X,03,2X,12,2X,
      1      A10,2X,15,2X,15,2X,15,2X,A30)
      CLOSE(REL1)
      TERMINATE
      END
End-of-record

```

Figure 5-7. Program RELATE (Sheet 1 of 4)

PROGRAM RELATE 76/176 OPT=0 FTN 5.1+538

```

1      PROGRAM RELATE
2
3      C THIS PROGRAM READS BY RELATION AND PRINTS GRADES FOR
4      C STUDENT ID 120-44-3760. THE FIRST READ IS RANDOM
5      C FOLLOWED BY A TEST FOR NULL RECORD. THE SECOND READ IS
6      C SEQUENTIAL FOLLOWED BY A TEST FOR CONTROL BREAK.
7      C
8      INTEGER STATBLK(11)
9      CHARACTER SAVEKEY*11
10     SUBSCHEMA(RELATION)
11     C$ LIST(ALL=0)
12     C$ LIST(ALL)
13     ** INVOKE
14     C$ LIST(ALL=0)
15     C$ LIST(ALL)
16     CALL DMLINQ(0002,DBF0001,10HRELATION ,10H
17     +10H ,0"76710464332261536703")
18     CALL DMLDBST(STATBLK,11)
19     PRIVACY(CFILE,PRIVACY='XX99')
20     CALL DMLPRV(1,1,0,0002,
21     +0"60","XX99","","")
22     OPEN(REL1,ERR=50)
23     CALL DMLOPNR(DBN0001,DBA0001,2HI0,*50 )
24     STATID='120-44-3760'
25     READ(REL1,KEY=STATID,ERR=50)
26     CALL DMLRLK(DBN0001,0001,00001,0001,1,0011,0,0000,00,STATID ,
27     + 0001,*50 )
28     SAVEKEY=STATID
29     IF(SAVEKEY.EQ.'JJJJJJJJJJ') GO TO 70
30     PRINT 20, CSTID,GRADE
31     FORMAT (1H0,9X,A11,3X,F4.1)
32     READ(REL1,ERR=50,END=70)
33     CALL DMLRL(DBN0001,0001,1,1,*50 ,*70 )
34     IF(STATBLK(7).NE.0) GO TO 70
35     GO TO 15
36     PRINT 60,STATBLK
37     FORMAT (1X,'STATUS BLOCK' /
38     1X,04,2X,I5,2X,03,2X,I2,2X,
39     1X,A10,2X,I5,2X,I5,2X,I5,2X,A30)
40     CLOSE(REL1)
41     CALL DMLCLSR(DBN0001,DBA0001)
42     TERMINATE
43     CALL DMLEND
44     END

```

Figure 5-7. Program RELATE (Sheet 2 of 4)

```

--VARIABLE MAP--(LO=A)
--NAME--ADDRESS--BLOCK-----PROPERTIES-----TYPE-----SIZE
CODE      OB /D0002AB/
COURS     2B /DB0002/
CSTID     1B /DB0002/
DATE      OB /D0002AB/
DBA0001   276B
DBF0001   16B /DB0000/
DBF0002   67B /DB0000/
DBJ0001   OB /DB0001/ EQV
DBSTAT    3B /DB0000/
DBS0001   15B /DB0000/
DBS0002   66B /DB0000/
DBT0001   61B /DB0000/
DBT0002   132B /DB0000/
GRADE     OB /D0002AA/

--NAME--ADDRESS--BLOCK-----PROPERTIES-----TYPE-----SIZE
DBI0002    OB /DB0002/ EQV
DBN0001   134B /DB0000/
DBREALM    OB /DB0000/
DBRELST    10B /DB0000/
DBRUID     7B /DB0000/
DBR0001    12B /DB0000/
DBR0002    63B /DB0000/
DBSCNAM     4B /DB0000/
IDENT      OB /DB0002/ EQV
MAJOR      1B /DB0001/
SAVEKEY    274B
STATBLK    261B
STID       OB /DB0001/ EQV
UNITS      OB /D0002AC/

CHAR*1
INTEGER
INTEGER
INTEGER
INTEGER
INTEGER
INTEGER
INTEGER
CHAR*14
CHAR*20
CHAR*11
INTEGER
CHAR*11
INTEGER

```

```

--PROCEDURES--(LO=A)
--NAME--TYPE-----ARGS-----CLASS-----
DMLCLSR    2 SUBROUTINE
DMLDBST    2 SUBROUTINE
DMLEND     0 SUBROUTINE
DMLINV     6 SUBROUTINE
DMLOPNR    4 SUBROUTINE

--NAME--TYPE-----ARGS-----CLASS-----
DMLPRV     8 SUBROUTINE
DMLRL      6 SUBROUTINE
LOCF       12 SUBROUTINE
GENERIC    1 INTRINSIC

```

```

--STATEMENT LABELS--(LO=A)
--LABEL--ADDRESS-----PROPERTIES-----DEF
10 *NO REFS* 120
15 67B 124
20 134B FORMAT 125
25 *NO REFS* 127

--LABEL--ADDRESS-----PROPERTIES-----DEF
50 103B
60 137B FORMAT 130
70 105B 131
135

```

```

--ENTRY POINTS--(LO=A)
--NAME--ADDRESS--ARGS--

```

```

RELATE 5B 0

```

```

--STATISTICS--

```

```

PROGRAM-UNIT LENGTH 301B = 193
SCM LABELLED COMMON LENGTH 152B = 106
SCM STORAGE USED 60700B = 25024
COMPILE TIME 0.057 SECONDS

```

Output From Program Execution

120-44-3760	2.0
120-44-3760	3.0
120-44-3760	4.0
120-44-3760	3.5
120-44-3760	3.5

Figure 5-7. Program RELATE (Sheet 4 of 4)

Control Statements

```
Job statement
USER statement
CHARGE statement
ATTACH(SSLIB)
DML(SB=SSLIB,LV=F5)
FTN5(I=DMLOUT)
LDSET(LIB=DMSLIB)
LG0.
End-of-record
```

Source Program

PROGRAM CHARGES

```
C
C THIS PROGRAM DEMONSTRATES SUBSCRIPTING AND THE DML
C REWRITE CAPABILITY. RELATION REL2 ASSOCIATES FILES
C STUDENT AND ACCOUNT. FILE ACCOUNT IS DEFINED IN THE
C SCHEMA AS HAVING REPEATING ITEMS. THIS PROGRAM PERFORMS A
C RANDOM READ TO SELECT STUDENT ID 197-11-2140 AND
C WRITE REAL VALUES INTO THREE FIELDS. A TOTAL IS CALCULATED
C AND PRINTED OUT ALONG WITH THE NAME OF THE STUDENT.
C
```

```
INTEGER STATBLK(11)
SUBSCHEMA(BURSAR)
INVOKE
CALL DMLDBST(STATBLK,11)
TOTAL=0
OPEN(REL2,ERR=70)
STID='197-11-2140'
READ(REL2,KEY=STID,ERR=70)
TUITION(1)=1400
LAB(1)=75
BOOKS(1)=146
REWRITE(ACCOUNT,ERR=70)
TOTAL=TUITION(1) + LAB(1) + BOOKS(1)
50 PRINT 60,NAME,TOTAL
60 FORMAT('0',9X,A30,3X,F8.2)
GO TO 90
70 PRINT 80,STATBLK
80 FORMAT (1X,'STATUS BLOCK' /
1 1X,04,2X,I5,2X,I3,2X,I2,2X,
1 A10,2X,I5,2X,I5,2X,I5,2X,A30)
90 CLOSE(REL2)
TERMINATE
END
End-of-record
```

Figure 5-8. Program CHARGES (Sheet 1 of 3)

PROGRAM CHARGES 76/176 OPT=0 FTN 5.1+538

```

1      PROGRAM CHARGES
2
3      C
4      C THIS PROGRAM DEMONSTRATES SUBSCRIPTING AND THE DML
5      C REWRITE CAPABILITY. RELATION REL2 ASSOCIATES FILES
6      C STUDENT AND ACCOUNT. FILE ACCOUNT IS DEFINED IN THE
7      C SCHEMA AS HAVING REPEATING ITEMS. THIS PROGRAM PERFORMS A
8      C RANDOM READ TO SELECT STUDENT ID 197-11-2140 AND
9      C WRITE REAL VALUES INTO THREE FIELDS. A TOTAL IS CALCULATED
10     C AND PRINTED OUT ALONG WITH THE NAME OF THE STUDENT.
11
12     C
13     C INTEGER STATBLK(11)
14     C SUBSCHEMA(BURSAR)
15     C LIST(CALL=0)
16     C LIST(ALL)
17     C INVOKE
18     C LIST(ALL=0)
19     C LIST(ALL)
20     C CALL DMLINV(0002,DBF0001,10HBURSAR ,10H
21     C +10H ,0"16643753141007716046")
22     C CALL DMLDBST(STATBLK,11)
23     C TOTAL=0
24     C OPEN(REL2,ERR=70)
25     C CALL DMLOPNR(DBN0001,DBA0001,2H10,*70 )
26     C STID='197-11-2140'
27     C READ(REL2,KEY=STID,ERR=70)
28     C CALL DMLRLK(DBN0001,0001,00001,00001,1,0011,0,0000,00,STID
29     C + 0001,*70 )
30     C TUITION(1)=1400
31     C LAB(1)=75
32     C BOOKS(1)=146
33     C REWRITE(ACCOUNT,ERR=70)
34     C CALL DMLREW(DBF0002,0,0002,00001,*70 )
35     C TOTAL=TUITION(1) + LAB(1) + BOOKS(1)
36     C PRINT 60,NAME,TOTAL
37     C FORMAT('0",9X,A30,3X,F8.2)
38     C GO TO 90
39     C PRINT 80,STATBLK
40     C FORMAT (1X,'STATUS BLOCK' /
41     C 1X,04,2X,I5,2X,03,2X,I2,2X,
42     C 1A10,2X,I5,2X,I5,2X,I5,2X,A30)
43     C CLOSE(REL2)
44     C CALL DMLCLSR(DBN0001,DBA0001)
45     C TERMINATE
46     C CALL DMLEND
47     C END

```

Figure 5-8. Program CHARGES (Sheet 2 of 3)

```

--VARIABLE MAP--(LO=A)
--NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
ADDR      4B /DB0001/      CHAR*20
ASTID      OB /DB0002/ EQV  CHAR*11
BOOKS     40B /D0002AA/    REAL      16
CITY       6B /DB0001/      CHAR*10
DBA0001    255B             INTEGER     3
DBF0001    16B /DB0000/      INTEGER    35
DBF0002    67B /DB0000/      INTEGER    35
DBR0002    63B /DB0000/      INTEGER     3
DBSCNAM     4B /DB0000/      INTEGER     3
DBSTAT     3B /DB0000/      INTEGER     3
DBS0001    15B /DB0000/      INTEGER     2
DBS0002    66B /DB0000/      INTEGER     2
DBT0001    61B /DB0000/      INTEGER    16
DBT0002    132B /DB0000/     REAL
LAB         20B /D0002AA/    REAL
--NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
DBI0001     OB /DB0001/ EQV  CHAR*1
DBI0002     OB /DB0002/ EQV  CHAR*1
DBN0001    134B /DB0000/    INTEGER     3
DBREALM     OB /DB0000/    INTEGER     3
DBRELST     OB /DB0000/    INTEGER     2
DBRUID       7B /DB0000/    INTEGER     2
DBR0001     12B /DB0000/    INTEGER     3
MISC        60B /D0002AA/    REAL     16
NAME        1B /DB0001/    CHAR*30
STATBLK     242B            INTEGER    11
STATE       7B /DB0001/    CHAR*2
STID        OB /DB0001/ EQV  CHAR*11
TOTAL      260B            REAL
TUITION     OB /D0002AA/    REAL
ZIP         7B /DB0001/    CHAR*5

--PROCEDURES--(LO=A)
--NAME--TYPE--ARGS--CLASS--
DMLCLSR     2 SUBROUTINE    DMLOPNR
DMLDBST     2 SUBROUTINE    DMLREW
DMLEND      0 SUBROUTINE    DMLRLK
DMLINV      6 SUBROUTINE    LOCF
--NAME--TYPE--ARGS--CLASS--
DMLOPNR     4 SUBROUTINE
DMLREW      5 SUBROUTINE
DMLRLK     12 SUBROUTINE
LOCF        1 INTRINSIC

--STATEMENT LABELS--(LO=A)
--LABEL--ADDRESS--PROPERTIES--DEF
50 *NO REFS* 133
60 135B      FORMAT        134
70 105B      FORMAT        136
80 140B      FORMAT        137
90 107B      FORMAT        141

--ENTRY POINTS--(LO=A)
--NAME--ADDRESS--ARGS--
CHARGES     5B 0

--STATISTICS--
PROGRAM-UNIT LENGTH      261B = 177
SCM LABELLED COMMON LENGTH 251B = 169
SCM STORAGE USED        60700B = 25024
COMPILE TIME            0.056 SECONDS

Output From Program Execution
CAREN NIELSON
1621.00

```

Control Statements

```
Job statement
USER statement
CHARGE statement
ATTACH(SSLIB)
DML(SB=SSLIB,LV=F5)
FTN5(I=DMLOUT)
LDSET(LIB=DMSLIB)
LGO.
End-of-record
```

Source Program

```
PROGRAM ADMIT
C
C THIS PROGRAM DEMONSTRATES THE USE OF SUBROUTINES.
C WHEN AN INCOMPLETE(I) CODE IS FOUND IN FILE CFIL,
C SUBROUTINE PRNTSUB IS CALLED TO PRINT THE STUDENT ID
C AND COURSE ID. NOTICE THAT THE SUBROUTINE REQUIRES
C A SUBSCHEMA AND INVOKE STATEMENT. THESE STATEMENTS
C ARE REQUIRED IN EVERY PROGRAM UNIT TO ENSURE THE SAME
C DATA IS BEING REFERENCED IN COMMON.
C
INTEGER STATBLK(11)
SUBSCHEMA(ADMISSIONS)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')
OPEN(CFILE,ERR=70)
10 READ(CFILE,ERR=70,END=100)
IF(CODE.EQ. 'I') CALL PRNTSUB
GO TO 10
70 PRINT 80,STATBLK
80 FORMAT ('X,'STATUS BLOCK' /
1 1X,04,2X,15,2X,03,2X,12,2X,
1 A10,2X,15,2X,15,2X,15,2X,A30)
100 CLOSE(CFILE)
TERMINATE
END
C
SUBROUTINE PRNTSUB
SUBSCHEMA(ADMISSIONS)
INVOKE
PRINT 50,STUDENT,CCID
50 FORMAT('0",9X,A11,3X,A6)
RETURN
END
End-of-record
```

Figure 5-9. Program ADMIT (Sheet 1 of 5)


```

1
2
3 PROGRAM ADMIT
4
5 C THIS PROGRAM DEMONSTRATES THE USE OF SUBROUTINES.
6 C WHEN AN INCOMPLETE(I) CODE IS FOUND IN FILE CFILF,
7 C SUBROUTINE PRINTSUB IS CALLED TO PRINT THE STUDENT ID
8 C AND COURSE ID. NOTICE THAT THE SUBROUTINE REQUIRES
9 C A SUBSCHEMA AND INVOKE STATEMENT. THESE STATEMENTS
10 C ARE REQUIRED IN EVERY PROGRAM UNIT TO ENSURE THE SAME
11 C DATA IS BEING REFERENCED IN COMMON.
12
13 INTEGER STATBLK(11)
14 SUBSCHEMA(ADMISSIONS)
15 LIST(ALL=0)
16 LIST(ALL)
17 INVOKE
18 LIST(ALL=0)
19 LIST(ALL)
20 CALL DMLINV(0002,DBF0001,10HADMISSIONS,10H
21 +10H
22 ,0"5606531337542307610")
23 CALL DMLDBST(STATBLK,11)
24 PRIVACY(CFILE,PRIVACY='XX99')
25 CALL DMLPRV(1,0,0002,
26 +0"60","XX99
27 "," " " ")
28 OPEN(CFILE,ERR=70)
29 CALL DMLOPN(DBF0002,0002,2H10,*70 )
30 READ(CFILE,ERR=70,END=100)
31 CALL DMLRD(DBF0002,0002,1,1,*70 ,*100 )
32 IF(CODE .EQ. 'I') CALL PRINTSUB
33 GO TO 10
34 PRINT 80,STATBLK
35 FORMAT (1X,'STATUS BLOCK' /
36 1X,04,2X,I5,2X,03,2X,I2,2X,
37 1X,04,2X,I5,2X,03,2X,I2,2X,
38 1X,04,2X,I5,2X,03,2X,I2,2X,
39 1X,04,2X,I5,2X,03,2X,I2,2X,
40 1X,04,2X,I5,2X,03,2X,I2,2X,
41 1X,04,2X,I5,2X,03,2X,I2,2X,
42 1X,04,2X,I5,2X,03,2X,I2,2X,
43 1X,04,2X,I5,2X,03,2X,I2,2X,
44 1X,04,2X,I5,2X,03,2X,I2,2X,
45 1X,04,2X,I5,2X,03,2X,I2,2X,
46 1X,04,2X,I5,2X,03,2X,I2,2X,
47 1X,04,2X,I5,2X,03,2X,I2,2X,
48 1X,04,2X,I5,2X,03,2X,I2,2X,
49 1X,04,2X,I5,2X,03,2X,I2,2X,
50 1X,04,2X,I5,2X,03,2X,I2,2X,
51 1X,04,2X,I5,2X,03,2X,I2,2X,
52 1X,04,2X,I5,2X,03,2X,I2,2X,
53 1X,04,2X,I5,2X,03,2X,I2,2X,
54 1X,04,2X,I5,2X,03,2X,I2,2X,
55 1X,04,2X,I5,2X,03,2X,I2,2X,
56 1X,04,2X,I5,2X,03,2X,I2,2X,
57 1X,04,2X,I5,2X,03,2X,I2,2X,
58 1X,04,2X,I5,2X,03,2X,I2,2X,
59 1X,04,2X,I5,2X,03,2X,I2,2X,
60 1X,04,2X,I5,2X,03,2X,I2,2X,
61 1X,04,2X,I5,2X,03,2X,I2,2X,
62 1X,04,2X,I5,2X,03,2X,I2,2X,
63 1X,04,2X,I5,2X,03,2X,I2,2X,
64 1X,04,2X,I5,2X,03,2X,I2,2X,
65 1X,04,2X,I5,2X,03,2X,I2,2X,
66 1X,04,2X,I5,2X,03,2X,I2,2X,
67 1X,04,2X,I5,2X,03,2X,I2,2X,
68 1X,04,2X,I5,2X,03,2X,I2,2X,
69 1X,04,2X,I5,2X,03,2X,I2,2X,
70 1X,04,2X,I5,2X,03,2X,I2,2X,
71 1X,04,2X,I5,2X,03,2X,I2,2X,
72 1X,04,2X,I5,2X,03,2X,I2,2X,
73 1X,04,2X,I5,2X,03,2X,I2,2X,
74 1X,04,2X,I5,2X,03,2X,I2,2X,
75 1X,04,2X,I5,2X,03,2X,I2,2X,
76 1X,04,2X,I5,2X,03,2X,I2,2X,
77 1X,04,2X,I5,2X,03,2X,I2,2X,
78 1X,04,2X,I5,2X,03,2X,I2,2X,
79 1X,04,2X,I5,2X,03,2X,I2,2X,
80 1X,04,2X,I5,2X,03,2X,I2,2X,
81 1X,04,2X,I5,2X,03,2X,I2,2X,
82 1X,04,2X,I5,2X,03,2X,I2,2X,
83 1X,04,2X,I5,2X,03,2X,I2,2X,
84 1X,04,2X,I5,2X,03,2X,I2,2X,
85 1X,04,2X,I5,2X,03,2X,I2,2X,
86 1X,04,2X,I5,2X,03,2X,I2,2X,
87 1X,04,2X,I5,2X,03,2X,I2,2X,
88 1X,04,2X,I5,2X,03,2X,I2,2X,
89 1X,04,2X,I5,2X,03,2X,I2,2X,
90 1X,04,2X,I5,2X,03,2X,I2,2X,
91 1X,04,2X,I5,2X,03,2X,I2,2X,
92 1X,04,2X,I5,2X,03,2X,I2,2X,
93 1X,04,2X,I5,2X,03,2X,I2,2X,
94 1X,04,2X,I5,2X,03,2X,I2,2X,
95 1X,04,2X,I5,2X,03,2X,I2,2X,
96 1X,04,2X,I5,2X,03,2X,I2,2X,
97 1X,04,2X,I5,2X,03,2X,I2,2X,
98 1X,04,2X,I5,2X,03,2X,I2,2X,
99 1X,04,2X,I5,2X,03,2X,I2,2X,
100 1X,04,2X,I5,2X,03,2X,I2,2X,
101 1X,04,2X,I5,2X,03,2X,I2,2X,
102 1X,04,2X,I5,2X,03,2X,I2,2X,
103 1X,04,2X,I5,2X,03,2X,I2,2X,
104 1X,04,2X,I5,2X,03,2X,I2,2X,
105 1X,04,2X,I5,2X,03,2X,I2,2X,
106 1X,04,2X,I5,2X,03,2X,I2,2X,
107 1X,04,2X,I5,2X,03,2X,I2,2X,
108 1X,04,2X,I5,2X,03,2X,I2,2X,
109 1X,04,2X,I5,2X,03,2X,I2,2X,
110 1X,04,2X,I5,2X,03,2X,I2,2X,
111 1X,04,2X,I5,2X,03,2X,I2,2X,
112 1X,04,2X,I5,2X,03,2X,I2,2X,
113 1X,04,2X,I5,2X,03,2X,I2,2X,
114 1X,04,2X,I5,2X,03,2X,I2,2X,
115 1X,04,2X,I5,2X,03,2X,I2,2X,
116 1X,04,2X,I5,2X,03,2X,I2,2X,
117 1X,04,2X,I5,2X,03,2X,I2,2X,
118 1X,04,2X,I5,2X,03,2X,I2,2X,
119 1X,04,2X,I5,2X,03,2X,I2,2X,
120 1X,04,2X,I5,2X,03,2X,I2,2X,
121 1X,04,2X,I5,2X,03,2X,I2,2X,
122 1X,04,2X,I5,2X,03,2X,I2,2X,
123 1X,04,2X,I5,2X,03,2X,I2,2X,
124 1X,04,2X,I5,2X,03,2X,I2,2X,
125 1X,04,2X,I5,2X,03,2X,I2,2X,
126 1X,04,2X,I5,2X,03,2X,I2,2X,
127 1X,04,2X,I5,2X,03,2X,I2,2X,
128 1X,04,2X,I5,2X,03,2X,I2,2X,
129 1X,04,2X,I5,2X,03,2X,I2,2X,
130 1X,04,2X,I5,2X,03,2X,I2,2X,
131 1X,04,2X,I5,2X,03,2X,I2,2X,
132 1X,04,2X,I5,2X,03,2X,I2,2X,
133 1X,04,2X,I5,2X,03,2X,I2,2X,
134 1X,04,2X,I5,2X,03,2X,I2,2X,
135 1X,04,2X,I5,2X,03,2X,I2,2X,
136 1X,04,2X,I5,2X,03,2X,I2,2X,
137 1X,04,2X,I5,2X,03,2X,I2,2X,
138 1X,04,2X,I5,2X,03,2X,I2,2X,
139 1X,04,2X,I5,2X,03,2X,I2,2X,
140 1X,04,2X,I5,2X,03,2X,I2,2X,
141 1X,04,2X,I5,2X,03,2X,I2,2X,
142 1X,04,2X,I5,2X,03,2X,I2,2X,
143 1X,04,2X,I5,2X,03,2X,I2,2X,
144 1X,04,2X,I5,2X,03,2X,I2,2X,
145 1X,04,2X,I5,2X,03,2X,I2,2X,
146 1X,04,2X,I5,2X,03,2X,I2,2X,
147 1X,04,2X,I5,2X,03,2X,I2,2X,
148 1X,04,2X,I5,2X,03,2X,I2,2X,
149 1X,04,2X,I5,2X,03,2X,I2,2X,
150 1X,04,2X,I5,2X,03,2X,I2,2X,
151 1X,04,2X,I5,2X,03,2X,I2,2X,
152 1X,04,2X,I5,2X,03,2X,I2,2X,
153 1X,04,2X,I5,2X,03,2X,I2,2X,
154 1X,04,2X,I5,2X,03,2X,I2,2X,
155 1X,04,2X,I5,2X,03,2X,I2,2X,
156 1X,04,2X,I5,2X,03,2X,I2,2X,
157 1X,04,2X,I5,2X,03,2X,I2,2X,
158 1X,04,2X,I5,2X,03,2X,I2,2X,
159 1X,04,2X,I5,2X
```

5-19

```

--VARIABLE MAP--(LO=A)
--NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
CCID      2B /DB0002/          CHAR*6
CID        0B /DB0001/          EQV
CODE       3B /DB0002/          CHAR*6
DBF0001    15B /DB0000/          INTEGER
DBF0002    66B /DB0000/          INTEGER
DBI0001     0B /DB0001/          EQV
DBI0002     0B /DB0002/          EQV
DBREALM     0B /DB0000/          EQV
DBRELST    10B /DB0000/          EQV
DBRUID       7B /DB0000/          EQV
DBR0001     11B /DB0000/          EQV
DBR0002     62B /DB0000/          EQV
DBSCNAM     4B /DB0000/          EQV

--NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
DBSTAT      3B /DB0000/          INTEGER
DBS0001     14B /DB0000/          INTEGER
DBS0002     65B /DB0000/          INTEGER
DBT0001     60B /DB0000/          INTEGER
DBT0002    131B /DB0000/          INTEGER
IDENT        0B /DB0002/          EQV
NAME         0B /DB0001/          CHAR*14
PREREQ       4B /DB0001/          CHAR*20
SCHOOL       2B /DB0001/          CHAR*6
STATBLK     177B /DB0001/          CHAR*20
STUDENT      1B /DB0002/          INTEGER
UNITS        0B /DB001AA/          CHAR*11

--NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
DBSCNAM     4B /DB0000/          EQV

--PROCEDURES--(LO=A)
--NAME--TYPE--ARGS--CLASS--
DMLCLS      2 SUBROUTINE
DMLDBST     2 SUBROUTINE
DMLEND       0 SUBROUTINE
DMLINV       6 SUBROUTINE
DMLOPN       4 SUBROUTINE

--STATEMENT LABELS--(LO=A)
--LABEL--ADDRESS--PROPERTIES--DEF
10 44B 106
70 60B 109
80 106B FORMAT 110
100 62B 114

--ENTRY POINTS--(LO=A)
--NAME--ADDRESS--ARGS--
ADMIT 5B 0

--STATISTICS--
PROGRAM-UNIT LENGTH 212B = 138
SCM LABELLED COMMON LENGTH 146B = 102
SCM STORAGE USED 60700B = 25024
COMPILE TIME 0.046 SECONDS

```

Figure 5-9. Program ADMIT (Sheet 3 of 5)

SUBROUTINE PRNTSUB 76/176 OPT=0

FTN 5.1+538

```

1      C
2      SUBROUTINE PRNTSUB
3      ** SUBSCHEMA(ADMISSIONS)
4      C$ LIST(ALL=0)
79     C$ LIST(ALL)
80     ** INVOKE
81     C$ LIST(ALL=0)
87     C$ LIST(ALL)
88     CALL DMLINV(0002,DBF0001,10HADMISSIONS,10H
89                ,0"56065313377542307610")
90     PRINT 50,STUDENT,CCID
91     FORMAT("0",9X,A11,3X,A6)
92     RETURN
93     END

```

---VARIABLE MAP---(LO=A)

NAME	ADDRESS	BLOCK	PROPERTIES	TYPE	SIZE
CCID	2B	/DB0002/		CHAR*6	
CID	0B	/DB0001/		EQV	
CODE	3B	/DB0002/		CHAR*1	
DBF0001	15B	/DB0000/		INTEGER	35
DBF0002	66B	/DB0000/		INTEGER	35
DBI0001	0B	/DB0001/		EQV	
DBI0002	0B	/DB0002/		EQV	
DBREALM	0B	/DB0000/		INTEGER	3
DBRELST	10B	/DB0000/		INTEGER	1
DBRUID	7B	/DB0000/		INTEGER	3
DBR0001	11B	/DB0000/		INTEGER	3
DBR0002	62B	/DB0000/		INTEGER	3
DBSCNAM	4B	/DB0000/		INTEGER	3
DBSTAT	3B	/DB0000/		INTEGER	
DBS0001	14B	/DB0000/		INTEGER	
DBS0002	65B	/DB0000/		INTEGER	
DBT0001	60B	/DB0000/		INTEGER	2
DBT0002	131B	/DB0000/		INTEGER	2
IDENT	0B	/DB0002/		EQV	
NAME	0B	/DB0001/		CHAR*14	
PREREQ	4B	/DB0001/		CHAR*20	
SCHOOL	2B	/DB0001/		CHAR*6	
STUDENT	1B	/DB0002/		CHAR*20	
UNITS	0B	/DB0002/		CHAR*11	
				INTEGER	

---PROCEDURES---(LO=A)

NAME	TYPE	ARGS	CLASS
DMLINV	GENERIC	6	SUBROUTINE
LOC		1	INTRINSIC

---STATEMENT LABELS---(LO=A)

LABEL	ADDRESS	PROPERTIES	DEF
50	44B	FORMAT	91

Figure 5-9. Program ADMIT (Sheet 4 of 5)

--ENTRY POINTS--(LO=A)
--NAME---ADDRESS--ARGS---

PRNTSUB 4B 0

--STATISTICS--

PROGRAM-UNIT LENGTH 67B = 55
SCM LABELLED COMMON LENGTH 146B = 102
SCM STORAGE USED 60700B = 25024
COMPILE TIME 0.033 SECONDS

Output From Program Execution

100-22-5860	PSY002
122-13-6704	PSY136
387-14-1232	MATH10
387-14-1232	CHM005
387-14-1232	PSY136
387-14-1232	BUS017
678-12-1144	HIS103

Figure 5-9. Program ADMIT (Sheet 5 of 5)

This user's guide assumes that CDCS is active and available for your job. This method of operation implies established schemas, appropriate sub-schema libraries, and successfully implemented applications. Any change in the data base environment, such as the addition of a new file definition, forces the data administrator to terminate CDCS and to reinitiate CDCS with a new master directory file attached. By using the CDCS Batch Test Facility, you can have CDCS running as a normal batch job. Each time you run your job, you attach a new version of the master directory file.

The CDCS Batch Test Facility is an absolute program called CDCSBTF. The program resides on the system library and is called into execution by the CDCSBTF control statement. The format of the CDCSBTF control statement is shown in figure 6-1.

CDCSBTF(lfn-1 [,lfn-2] ...)

lfn Specifies the logical file name of a relocatable binary file containing a user program. Up to 16 files can be specified.

Figure 6-1. CDCSBTF Control Statement Format

When CDCS is executing as the Batch Test Facility, the name of the output file produced by CDCS is CDCSOUT.

REQUIREMENTS

When you are running application programs with the CDCSBTF program, you must meet certain requirements. These requirements are:

- Attach the master directory file, which must have the local file name MSTRDIR.
- Attach any necessary log files. It is the responsibility of the data administrator to provide you information about the log files.
- Have the application program in relocatable binary format as a local or a permanent file.
- Assign unique names to non-CDCS files. Do not use any of the following names:

INPUT
OUTPUT
MSTRDIR
CDCSOUT
Pnnnnnn } where n is any six digits
Xnnnnnn }
Fnnnnnn }

A name beginning with ZZZZZ
A log file name

- Be sure your application program executes a DML TERMINATE statement before a FORTRAN STOP or END statement. Failure to do this would discontinue processing for all programs specified in the CDCSBTF control statement that have not completed execution.
- Set the DB parameter in the FTN5 control statement equal to 0. Multiple copies of CDCSBTF can be run concurrently, and as many as 16 user programs can be run with a copy of CDCSBTF. If more than two concurrent calls to the RECOVER routine are made, CDCS aborts processing.

OBTAINING LOAD MAPS

You can obtain load maps by setting sense switches 1 through 4 prior to execution of the CDCSBTF control statement. Each sense switch setting corresponds to different information on the load map. The settings and the associated types of information are shown in table 6-1.

TABLE 6-1. LOAD MAP SWITCH SETTINGS

Setting	Load Map Information
SWITCH(1)	Statistics (S)
SWITCH(2)	Block maps (B)
SWITCH(3)	Entry point maps (EO)
SWITCH(4)	Entry point cross-reference maps (X).

EXECUTING THE CDCS BATCH TEST FACILITY

You need to include a number of control statements when executing the CDCS Batch Test Facility for your FORTRAN application program. Figure 6-2 provides a sample list of statements. Parameters correspond to the application that appears in appendix C.

<u>NOS Operating System</u>	<u>NOS/BE Operating System</u>	
Jobname(CMxxxxx)	Jobname(CMxxxxx)	Names the job and specifies maximum field length.
USER statement		Identifies the user.
CHARGE statement	ACCOUNT statement	Specifies the account to which the job's use of system resources is logged.
ATTACH(SSLIB/UN=xxx)	ATTACH(SSLIB,ID=xxx)	Attaches the sub-schema.
DML(SB=SSLIB,LV=F5)	DML(SB=SSLIB,LV=F5)	Preprocesses the DML statements in the FORTRAN program.
FTN5(I=DMLOUT,DB=0)	FTN5(I=DMLOUT,DB=0)	Compiles the FORTRAN program and places it on the file LGO.
ATTACH(MSTRDIR/UN=xxx)	ATTACH(MSTRDIR,ID=xxx)	Attaches the master directory.
ATTACH(LOGFILE/UN=xxx)	ATTACH(LOGFILE,ID=xxx)	Attaches the journal log file.
SWITCH(1)	SWITCH(1)	Requests statistics on program-initiated load.
SWITCH(2)	SWITCH(2)	Requests block map on program-initiated load.
SWITCH(3)	SWITCH(3)	Requests entry point map on program-initiated load.
SWITCH(4)	SWITCH(4)	Requests entry point cross-reference map on program-initiated load.
LIBRARY(DMSLIB)	LIBRARY(DMSLIB)	Specifies that the library DMSLIB is to be used to satisfy externals.
CDCSBTF(LGO)	CDCSBTF(LGO)	Executes CDCSBTF.
REWIND(CDCSOUT)	REWIND(CDCSOUT)	Rewinds the CDCS output file.
COPY(CDCSOUT,OUTPUT)	COPY(CDCSOUT,OUTPUT)	Prints the CDCS output file.
CRMEP.	CRMEP.	Prints the CRM error file.
EXIT.	EXIT.	Establishes processing if error occurs.
DMP.	DMP.	Dumps the exchange package.
DMP(177000)	DMP(177000)	Dumps the contents of the field length.
CRMEP.	CRMEP.	Prints the CRM error file.

Figure 6-2. Sample FORTRAN Execution of CDCS Batch Test Facility

STANDARD CHARACTER SETS

A

Control Data operating systems offer the following variations of a basic character set:

- CDC 64-character set
- CDC 63-character set
- ASCII 64-character set
- ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed or (for NOS only) dead started.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through

the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified also by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described previously for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table (table A-1) are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

Standard collating sequences for the two printer character sets are shown in tables A-2 and A-3.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC			ASCII		
	Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(0-8-4	34	(12-8-5	050
52)	12-8-4	74)	11-8-5	051
53	\$	11-8-3	53	\$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	#	0-8-6	36	#	8-3	043
61	[8-7	17	[12-8-2	133
62]	0-8-2	32]	11-8-2	135
63	%††	8-6	16	%††	0-8-4	045
64	"	8-4	14	" (quote)	8-7	042
65	_	0-8-5	35	_ (underline)	0-8-5	137
66	!	11-0	52	!	12-8-7	041
67	&	0-8-7	37	&	12	046
70	' (apostrophe)	11-8-5	55	' (apostrophe)	8-5	047
71	?	11-8-6	56	?	0-8-7	077
72	<	12-0	72	<	12-8-4	074
73	>	11-8-7	57	>	0-8-6	076
74	@	8-5	15	@	8-4	100
75	\	12-8-5	75	\	0-8-2	134
76	~ (circumflex)	12-8-6	76	~ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

† Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

†† In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55g).

TABLE A-2. CDC CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		CDC Graphic	Display Code	External BCD	Collating Sequence Decimal/Octal		CDC Graphic	Display Code	External BCD
00	00	blank	55	20	32	40	H	10	70
01	01	<	74	15	33	41	I	11	71
02	02	%	63 †	16 †	34	42	v	66	52
03	03	[61	17	35	43	J	12	41
04	04	→	65	35	36	44	K	13	42
05	05	≡	60	36	37	45	L	14	43
06	06	^	67	37	38	46	M	15	44
07	07	↑	70	55	39	47	N	16	45
08	10	↓	71	56	40	50	O	17	46
09	11	>	73	57	41	51	P	20	47
10	12	>	75	75	42	52	Q	21	50
11	13]	76	76	43	53	R	22	51
12	14	.	57	73	44	54]	62	32
13	15)	52	74	45	55	S	23	22
14	16	:	77	77	46	56	T	24	23
15	17	+	45	60	47	57	U	25	24
16	20	\$	53	53	48	60	V	26	25
17	21	*	47	54	49	61	W	27	26
18	22	-	46	40	50	62	X	30	27
19	23	/	50	21	51	63	Y	31	30
20	24	,	56	33	52	64	Z	32	31
21	25	(51	34	53	65	:	00 †	none†
22	26	=	54	13	54	66	0	33	12
23	27	≠	64	14	55	67	1	34	01
24	30	<	72	72	56	70	2	35	02
25	31	A	01	61	57	71	3	36	03
26	32	B	02	62	58	72	4	37	04
27	33	C	03	63	59	73	5	40	05
28	34	D	04	64	60	74	6	41	06
29	35	E	05	65	61	75	7	42	07
30	36	F	06	66	62	76	8	43	10
31	37	G	07	67	63	77	9	44	11

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.

TABLE A-3. ASCII CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		ASCII Graphic Subset	Display Code	ASCII Code	Collating Sequence Decimal/Octal		ASCII Graphic Subset	Display Code	ASCII Code
00	00	blank	55	20	32	40	@	74	40
01	01	!	66	21	33	41	A	01	41
02	02	"	64	22	34	42	B	02	42
03	03	#	60	23	35	43	C	03	43
04	04	\$	53	24	36	44	D	04	44
05	05	%	63†	25	37	45	E	05	45
06	06	&	67	26	38	46	F	06	46
07	07	'	70	27	39	47	G	07	47
08	10	(51	28	40	50	H	10	48
09	11)	52	29	41	51	I	11	49
10	12	*	47	2A	42	52	J	12	4A
11	13	+	45	2B	43	53	K	13	4B
12	14	,	56	2C	44	54	L	14	4C
13	15	-	46	2D	45	55	M	15	4D
14	16	.	57	2E	46	56	N	16	4E
15	17	/	50	2F	47	57	O	17	4F
16	20	0	33	30	48	60	P	20	50
17	21	1	34	31	49	61	Q	21	51
18	22	2	35	32	50	62	R	22	52
19	23	3	36	33	51	63	S	23	53
20	24	4	37	34	52	64	T	24	54
21	25	5	40	35	53	65	U	25	55
22	26	6	41	36	54	66	V	26	56
23	27	7	42	37	55	67	W	27	57
24	30	8	43	38	56	70	X	30	58
25	31	9	44	39	57	71	Y	31	59
26	32	:	00†	3A	58	72	Z	32	5A
27	33	;	77	3B	59	73	[61	5B
28	34	<	72	3C	60	74	\	75	5C
29	35	=	54	3D	61	75]	62	5D
30	36	>	73	3E	62	76	^	76	5E
31	37	?	71	3F	63	77	_	65	5F

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

Access Control -

Protection of data from unauthorized access or modification.

Actual Key -

A file organization in which records are stored according to their system-assigned key values.

Advanced Access Methods (AAM) -

A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor. See CYBER Record Manager.

Alias -

A data name used in the sub-schema in place of a schema data name.

Area -

A uniquely named schema data base subdivision that contains data records; identified in the sub-schema as a realm; a file.

Basic Access Methods (BAM) -

A file manager that processes sequential and word addressable file organizations. See CYBER Record Manager.

CDCS -

See CYBER Database Control System.

CDCS Batch Test Facility -

A facility that allows an application to simulate a data base environment without impacting any other CDCS users on the system.

Child Record Occurrence -

For relation processing, a record occurrence that has another record occurrence (the parent record occurrence) at the next numerically lower rank in the relation; for constraint processing, a record occurrence that is the dependent member of a condition defined by a constraint.

Common Item -

A data item that appears in two or more files joined in a relation; in each instance, the data item contains the same value.

Concurrency -

Simultaneous access to the same data in a data base by two or more application programs during a given span of time.

Constraint -

A control imposed on records in related files or on items in a single file for the purpose of protecting the integrity of data in a data base during update operations. A constraint is defined in the schema and is based on the common item in the records.

Control Break -

A condition that occurs during a relation read to signify a new record occurrence was read for the parent file.

CRM -

See CYBER Record Manager.

CYBER Database Control System (CDCS) -

The controlling module that provides the interface between the application program and the data base.

CYBER Record Manager (CRM) -

A generic term relating to the common products BAM and AAM, which run under the NOS and NOS/BE operating systems and allow a variety of record types, block types, and file organizations to be created and accessed. The execution time input/output of the DMS-170 products is implemented through CRM. All CRM file processing requests ultimately pass through the operating system input/output routines.

Data Administrator -

A person who defines the format and organization of the data base.

Data Base -

A systematically organized, central pool of information; organization is described by a schema.

Data Base Procedure -

A special-purpose routine that performs a predefined operation; specified in the schema and initiated by CDCS.

Data Base Status Block -

An array defined within an application program to which CDCS returns information concerning the status of operations on data base files and relations. The status block is updated after each CDCS operation.

Data Description Language (DDL) -

The language used to structure the schema and the sub-schema.

Data Item -

A unit of data within a record; can be a variable or an array in the FORTRAN sub-schema.

Data Manipulation Language (DML) -

The extensions to FORTRAN that provide access to a DMS-170 data base.

DDL -

See Data Description Language.

Deadlock -

A situation that arises in concurrent data base access when two or more application programs, each with locked resources, are contending for a resource that is locked by one of the other application programs, and none of the programs can proceed without that resource.

Direct Access -

In the context of CRM, one of the five file organizations. The organization is characterized by the system hashing of the unique key within each file record to distribute records randomly in blocks called home blocks of the file.

In the context of NOS permanent files, a file that is accessed and modified directly, as contrasted with an indirect access permanent file.

DML -

See Data Manipulation Language.

File -

A collection of records treated as a unit; an area in the schema; a realm in the sub-schema.

Hierarchical Tree Structure -

A representation that commonly illustrates record occurrences for files joined in a directed relation. The root of the tree is a record occurrence in the root file, and each successive level represents the record occurrences in each joined file.

Home Block -

Mass storage allocated for a file with direct access organization at the time the file is created.

Indexed Sequential -

A file organization in which records are stored in ascending order by key.

Keyword -

A word that is required in a source program statement.

Log Files -

Files that hold historical records of operations performed by users on data base areas.

Mapping -

The process by which CDCS produces a record or item image conforming to the schema or sub-schema description.

Master Directory -

A file created by the data administrator and used by CDCS in processing. This information consists of schema and sub-schema tables, media parameters, and data base procedure library and logging specifications.

Multiple-Index Processor -

A processor that allows AAM files to be accessed by alternate keys.

Null Record Occurrence -

A record occurrence composed of the display code right bracket symbol in each character position. The null record occurrence is used in a relation occurrence to denote that no record occurrence qualifies or that a record occurrence does not exist at a given level in the relation.

Parent Record Occurrence -

For relation processing, a record occurrence that has another record occurrence at the next numerically higher rank in the relation; for constraint processing, a record occurrence that is the dominant member of a condition defined by a constraint.

Permanent File -

A file that resides on a mass storage permanent file device and can be retained for longer than a single job. The file is protected against accidental destruction and can be protected against unauthorized access.

Privacy Key -

A character constant, variable name, or unsubscripted name that is included in a FORTRAN DML PRIVACY statement to gain access to a particular realm.

Rank -

The rank of a file in a relation corresponds to the position of the file in the schema definition of the relation. The ranks of the files joined in a relation are numbered consecutively, with the root file having a rank of 1.

Realm -

A uniquely named sub-schema data base subdivision that contains data records; identified in the schema as an area; a file.

Realm Ordinal -

A unique identifier assigned to each realm in a sub-schema when the sub-schema is compiled. Sub-schema realm ordinals are used in conjunction with the data base status block.

Record -

A named collection of one or more data items that are treated as a unit.

Record Occurrence -

An actual data base record that conforms to a record description in the schema.

Record Type -

The description of the attributes of a record; record layout.

Recovery -

A process that makes a data base useful after some type of software or hardware failure has occurred.

Recovery Point -

A user-declared or system-generated point to which CDCS guarantees recovery with no loss of data. User-declared recovery points are generated by FORTRAN calls to DMLRPT.

Relation -

A group of files that are related by common data items; therefore the files can be opened, closed, or read by a single request. Relations are defined in the schema.

Relation Occurrence -

The logical concatenation of a record occurrence from each record type specified in the relation.

Restriction -

Criteria that must be satisfied by a record occurrence in a relation before it can be made available to the application program. Restrictions are defined in the sub-schema.

Root Realm -

The first realm listed in a relation; the root realm has the rank of 1 in a relation; record occurrences of the root realm are pictured as the root of a tree in a hierarchical tree structure.

Schema -

A detailed description of the internal structure of the complete data base.

Status Block -

See Data Base Status Block.

Sub-Schema -

A detailed description of the portion of the data base to be made available to one or more application programs.

Sub-Schema Item Ordinal -

An identifier, unique within a record, assigned to each item in a sub-schema when the sub-schema is compiled. Sub-schema item ordinals are used in conjunction with the data base status block.

Sub-Schema Library -

A permanent file containing one or more sub-schemas.

O

O

O

O

O

O

O

This appendix contains the source programs and control statements used to generate the data base environment for the university application presented in this user's guide. Although all programs reflect operation under the NOS operating system, conversion to the NOS/BE operating system could be accomplished by making the following changes:

- Substitute a NOS/BE ACCOUNT control statement for the NOS USER and CHARGE control statements.
- Substitute the NOS/BE REQUEST and CATALOG control statements for the NOS DEFINE control statement.
- Substitute the NOS/BE file identification parameter ID for the NOS file identification parameter UN. This substitution applies to the source input for the master directory and to the Query Update program.

Setting up a DMS-170 data management environment is a data administrator responsibility; the process is shown here, however, to allow the reader to duplicate the application and use it to gain an understanding of DMS-170 FORTRAN application programming. The source input for the jobs shown in this appendix illustrates the university application being created by a series of batch jobs. The source input for each job is shown exactly as required for processing on NOS with two exceptions:

- End-of-record is indicated by the statement end-of-record and a blank line that is inserted to improve readability of the text.
- End of input for the job is indicated by the statement end-of-information.

The steps the data administrator takes to establish the application are listed in appropriate order as follows:

1. Design, write, compile, and store the schema definition as a permanent file. A schema named UNIVERSITY is stored as a permanent file named UNIVERS. See figure C-1.
2. Design, write, compile, and store sub-schema definitions as a permanent file library. A FORTRAN sub-schema library is stored as a permanent file named SSLIB. Input consists of four separate sub-schemas (AVERAGE, RELATION, ADMISSIONS, BURSAR); the sub-schemas follow each other with no intervening end-of-records. See figure C-2 for both source input and the listing that results from compilation.
3. Design, write, compile, and store a sub-schema for use by a data base creation program. This application selects Query Update as the language to store the data. A Query Update sub-schema named CREATES is stored as a permanent file library named QULIB. See figure C-3.
4. Generate a master directory through the DBMSTRD utility. A master directory is stored as a permanent file named MSTRDIR. See figure C-4.
5. Write a program to store the data base. A Query Update program creates four data base files (COURSE, STUDENT, CURRICULUM, ACCOUNTING) using the Query Update sub-schema, and defines the appropriate index files assigned in the master directory. See figure C-5. (CDCS must be active to use this program.) For processing on NOS/BE, the series of control statements that must be executed for each area and index file before the QU control statement is executed is as follows: REQUEST, REWIND, CATALOG, and RETURN.
6. Establish CDCS as an active system. This must be done by the data administrator; the process is not shown in this guide.

```

Job statement
USER statement
CHARGE statement
FILE(PROFESS,F0=IS,XN=PNDX)
FILE(COURSE,F0=IS,XN=CRSNDX)
FILE(STUDENT,F0=IS,XN=SDNX)
FILE(CURRICU,F0=IS,XN=CRNDX)
FILE(ACCOUNT,F0=IS)
DEFINE(UNIVERS=UNIVERS/CT=PU,M=R)
DDL3(DS,SC=UNIVERS)
End-of-record

```

SCHEMA NAME IS UNIVERSITY.

AREA NAME IS PROFESSOR.
RECORD IS PROF-REC WITHIN PROFESSOR.

PROF-ID	TYPE CHARACTER 8.
PROF-NAME	PICTURE "X(30)".
ACADEMIC-FIELD	TYPE CHARACTER 20.

AREA NAME IS COURSE.
RECORD IS COURSE-REC WITHIN COURSE.

COURSE-ID	TYPE CHARACTER 6.
COURSE-NAME	PICTURE "X(20)".
SCHOOL	PICTURE "X(20)".
PROF-ID	TYPE CHARACTER 8.
PREREQUISITE	TYPE CHARACTER 6.
UNITS	TYPE DECIMAL.

AREA NAME IS STUDENT.
RECORD IS STUDENT-REC WITHIN STUDENT.

STUDENT-ID	TYPE CHARACTER 11.
STUDENT-NAME	PICTURE "X(30)".
STREET-ADDRESS	PICTURE "X(20)".
CITY	PICTURE "X(10)".
STATE	PICTURE "A(2)".
ZIP-CODE	PICTURE "X(5)".
PHONE	PICTURE "X(12)".
MAJOR	TYPE CHARACTER 20.

AREA NAME IS CURRICULUM
ACCESS-CONTROL LOCK IS "XX99".
RECORD IS CURR-REC WITHIN CURRICULUM.

IDENT	TYPE CHARACTER 14.
STUDENT-ID	TYPE CHARACTER 11.
COURSE-ID	TYPE CHARACTER 6.
GRADE	TYPE FLOAT
	CHECK VALUE 0.0 THRU 4.0.
COMPLETE-CODE	TYPE CHARACTER 1.
COMPLETE-DATE	TYPE CHARACTER 8.
UNITS	TYPE DECIMAL.

AREA NAME IS ACCOUNTING.
RECORD IS ACCT-REC WITHIN ACCOUNTING.

01 STUDENT-ID	TYPE CHARACTER 11.
01 TUITION	TYPE FLOAT OCCURS 16 TIMES.
01 LAB-FEES	TYPE FLOAT OCCURS 16 TIMES.
01 BOOKS	TYPE FLOAT OCCURS 16 TIMES.
01 MISC-FEES	TYPE FLOAT OCCURS 16 TIMES.

DATA CONTROL.

AREA NAME IS PROFESSOR
KEY IS PROF-ID OF PROF-REC
DUPLICATES ARE NOT ALLOWED
KEY IS ALTERNATE ACADEMIC-FIELD
DUPLICATES ARE ALLOWED.

AREA NAME IS COURSE
KEY IS COURSE-ID OF COURSE-REC
DUPLICATES ARE NOT ALLOWED
KEY IS ALTERNATE PROF-ID OF COURSE-REC
DUPLICATES ARE ALLOWED.

AREA NAME IS STUDENT
KEY IS STUDENT-ID OF STUDENT-REC
DUPLICATES ARE NOT ALLOWED
KEY IS ALTERNATE MAJOR
DUPLICATES ARE ALLOWED.

AREA NAME IS CURRICULUM
KEY IS IDENT
KEY IS ALTERNATE STUDENT-ID OF CURR-REC
DUPLICATES ARE ALLOWED
KEY IS ALTERNATE COURSE-ID OF CURR-REC
DUPLICATES ARE ALLOWED
KEY IS ALTERNATE GRADE
DUPLICATES ARE ALLOWED.

AREA NAME IS ACCOUNTING
KEY IS STUDENT-ID OF ACCT-REC
DUPLICATES ARE NOT ALLOWED.

CONSTRAINT NAME IS CON1
STUDENT-ID OF CURR-REC DEPENDS ON
STUDENT-ID OF STUDENT-REC.

CONSTRAINT NAME IS CON2
STUDENT-ID OF ACCT-REC DEPENDS ON
STUDENT-ID OF STUDENT-REC.

CONSTRAINT NAME IS CON3
COURSE-ID OF CURR-REC DEPENDS ON
COURSE-ID OF COURSE-REC.

RELATION NAME IS REL1
JOIN WHERE STUDENT-ID OF STUDENT-REC
EQ STUDENT-ID OF CURR-REC.

RELATION NAME IS REL2
JOIN WHERE STUDENT-ID OF STUDENT-REC
EQ STUDENT-ID OF ACCT-REC.

RELATION NAME IS REL3
JOIN WHERE PROF-ID OF PROF-REC
EQ PROF-ID OF COURSE-REC
COURSE-ID OF COURSE-REC
EQ COURSE-ID OF CURR-REC.

End-of-record

End-of-information

Figure C-1. The UNIVERSITY Schema

Source Input

```
Job statement
USER statement
CHARGE statement
ATTACH(UNIVERS)
DEFINE(SSLIB/CT=PU,M=W)
DDL(F5,SB=SSLIB,SC=UNIVERS)
End-of-record

SUBSCHEMA AVERAGE,SCHEMA=UNIVERSITY

ALIAS(REALM) CFILE=CURRICULUM
ALIAS(RECORD) CRECORD=CURR-REC
ALIAS(ITEM) STUDENT=STUDENT-ID.CURR-REC
ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC

REALM CFILE

RECORD CRECORD

CHARACTER*14 IDENT
CHARACTER*11 STUDENT
CHARACTER*6 COURSE
REAL GRADE
END

SUBSCHEMA COMPARE,SCHEMA=UNIVERSITY

ALIAS(REALM) PFILE=PROFESSOR
ALIAS(RECORD) PRECORD=PROF-REC
ALIAS(ITEM) PROFID=PROF-ID.PROF-REC
ALIAS(ITEM) PNAME=PROF-NAME

ALIAS(REALM) CRSFILE=COURSE
ALIAS(RECORD) CRSREC=COURSE-REC
ALIAS(ITEM) CRSID=COURSE-ID.COURSE-REC
ALIAS(ITEM) CRSNAME=COURSE-NAME
ALIAS(ITEM) PROF=PROF-ID.COURSE-REC
ALIAS(ITEM) FIELD=ACADEMIC-FIELD

ALIAS(REALM) CFILE=CURRICULUM
ALIAS(RECORD) CRECORD=CURR-REC
ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC
ALIAS(ITEM) CODE=COMPLETE-CODE
ALIAS(ITEM) DATE=COMPLETE-DATE

REALM PFILE
REALM CRSFILE
REALM CFILE

RECORD PRECORD
CHARACTER*8 PROFID
CHARACTER*30 PNAME
CHARACTER*20 FIELD

RECORD CRSREC
CHARACTER*6 CRSID
CHARACTER*20 CRSNAME
CHARACTER*8 PROF

RECORD CRECORD
CHARACTER*14 IDENT
CHARACTER*6 COURSE
CHARACTER*1 CODE
CHARACTER*8 DATE
REAL GRADE

RELATION REL3
RESTRICT CRECORD (CODE .EQ. 'C')
END
```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 1 of 8)

```

SUBSCHEMA RELATION,SCHEMA=UNIVERSITY

ALIAS(REALM)  SFIL=STUDENT
ALIAS(RECORD) SRECORD=STUDENT-REC
ALIAS(ITEM)   STID=STUDENT-ID.STUDENT-REC

ALIAS(REALM)  CFIL=CURRICULUM
ALIAS(RECORD) CRECORD=CURR-REC
ALIAS(ITEM)   CSTID=STUDENT-ID.CURR-REC
ALIAS(ITEM)   COUR=COURSE-ID.CURR-REC
ALIAS(ITEM)   PROF=PROF-NAME
ALIAS(ITEM)   CODE=COMPLETE-CODE
ALIAS(ITEM)   DATE=COMPLETE-DATE

REALM SFIL
REALM CFIL

RECORD SRECORD
CHARACTER*11 STID
CHARACTER*20 MAJOR

RECORD CRECORD
CHARACTER*14 IDENT
CHARACTER*11 CSTID
CHARACTER*6  COUR
REAL GRADE
CHARACTER CODE
CHARACTER*8 DATE
INTEGER UNITS
RELATION REL1
RESTRICT CRECORD(CODE.EQ.'C')
END

SUBSCHEMA ADMISSIONS,SCHEMA=UNIVERSITY

ALIAS(RECORD) CRSREC=COURSE-REC
ALIAS(ITEM)   CID=COURSE-ID.COURSE-REC
ALIAS(ITEM)   NAME=COURSE-NAME
ALIAS(ITEM)   PREREQ=PREREQUISITE

ALIAS(REALM)  CFIL=CURRICULUM
ALIAS(RECORD) CURREC=CURR-REC
ALIAS(ITEM)   STUDENT=STUDENT-ID
ALIAS(ITEM)   CCID=COURSE-ID.CURR-REC
ALIAS(ITEM)   CODE=COMPLETE-CODE

REALM COURSE
REALM CFIL

RECORD CRSREC
CHARACTER*6 CID
CHARACTER*20 NAME
CHARACTER*20 SCHOOL
CHARACTER*6 PREREQ
INTEGER UNITS

RECORD CURREC
CHARACTER*14 IDENT
CHARACTER*11 STUDENT
CHARACTER*6 CCID
CHARACTER CODE
END

```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 2 of 8)

SUBSCHEMA BURSAR,SCHEMA=UNIVERSITY

ALIAS(RECORD) STREC=STUDENT-REC
ALIAS(ITEM) STID=STUDENT-ID.STUDENT-REC
ALIAS(ITEM) NAME=STUDENT-NAME
ALIAS(ITEM) ADDR=STREET-ADDRESS
ALIAS(ITEM) ZIP=ZIP-CODE

ALIAS(REALM) ACCOUNT=ACCOUNTING
ALIAS(RECORD) ACCTREC=ACCT-REC
ALIAS(ITEM) ASTID=STUDENT-ID.ACCT-REC
ALIAS(ITEM) LAB=LAB-FEES
ALIAS(ITEM) MISC=MISC-FEES

REALM STUDENT
REALM ACCOUNT

RECORD STREC
CHARACTER*11 STID
CHARACTER*30 NAME
CHARACTER*20 ADDR
CHARACTER*10 CITY
CHARACTER*2 STATE
CHARACTER*5 ZIP

RECORD ACCTREC
CHARACTER*11 ASTID
REAL TUITION(16)
REAL LAB(16)
REAL BOOKS(16)
REAL MISC(16)

RELATION REL2
END

End-of-record

End-of-information

Compilation Source Listings

AVERAGE

* SOURCE LISTING * (80351) DDLF 1.2+538.

	00001	SUBSCHEMA AVERAGE,SCHEMA=UNIVERSITY
	00002	
	00003	ALIAS(REALM) CFILE=CURRICULUM
	00004	ALIAS(RECORD) CRECORD=CURR-REC
	00005	ALIAS(ITEM) STUDENT=STUDENT-ID.CURR-REC
	00006	ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC
	00007	
	00008	REALM CFILE
	00009	
	00010	RECORD CRECORD
	00011	
** WITHIN CFILE		
	00012	CHARACTER*14 IDENT
** ORDINAL 1	00013	CHARACTER*11 STUDENT
** ORDINAL 2	00014	CHARACTER*6 COURSE
** ORDINAL 3	00015	REAL GRADE
** ORDINAL 4	00016	END

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 3 of 8)

```

00017
*****
PRIMARY KEY 00012 IDENT FOR AREA CFILE
ALTERNATE KEY 00013 STUDENT FOR AREA CFILE
ALTERNATE KEY 00014 COURSE FOR AREA CFILE
ALTERNATE KEY 00015 GRADE FOR AREA CFILE
*****
RECORD MAPPING IS NEEDED FOR REALM - CFILE

```

```

-----
SUBSCHEMA AVERAGE CHECKSUM
35514310376143061021
-----
DDLF COMPLETE. 0 DDLF.
476008 CM USED. 0.064 CP SECS.

```

```

COMPARE * SOURCE LISTING * (80351) DDLF 1.2+538.

00001 SUBSCHEMA COMPARE, SCHEMA=UNIVERSITY
00002
00003 ALIAS(REALM) PFILE=PROFESSOR
00004 ALIAS(RECORD) PRECORD=PROF-REC
00005 ALIAS(ITEM) PROFID=PROF-ID.PROF-REC
00006 ALIAS(ITEM) PNAME=PROF-NAME
00007
00008 ALIAS(REALM) CRSFILE=COURSE
00009 ALIAS(RECORD) CRSREC=COURSE-REC
00010 ALIAS(ITEM) CRSID=COURSE-ID.COURSE-REC
00011 ALIAS(ITEM) CRSNAME=COURSE-NAME
00012 ALIAS(ITEM) PROF=PROF-ID.COURSE-REC
00013 ALIAS(ITEM) FIELD=ACADEMIC-FIELD
00014
00015 ALIAS(REALM) CFILE=CURRICULUM
00016 ALIAS(RECORD) CRECORD=CURR-REC
00017 ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC
00018 ALIAS(ITEM) CODE=COMPLETE-CODE
00019 ALIAS(ITEM) DATE=COMPLETE-DATE
00020
00021 REALM PFILE
00022 REALM CRSFILE
00023 REALM CFILE
00024
00025 RECORD PRECORD
** WITHIN PFILE
00026 CHARACTER*8 PROFID
** ORDINAL 1
00027 CHARACTER*30 PNAME
** ORDINAL 2
00028 CHARACTER*20 FIELD
00029
** ORDINAL 3
00030 RECORD CRSREC
** WITHIN CRSFILE
00031 CHARACTER*6 CRSID
** ORDINAL 1
00032 CHARACTER*20 CRSNAME
** ORDINAL 2
00033 CHARACTER*8 PROF
00034
** ORDINAL 3

```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 4 of 8)

```

00035          RECORD CRECORD
** WITHIN CFILE
00036          CHARACTER*14 IDENT
** ORDINAL 1
00037          CHARACTER*6 COURSE
** ORDINAL 2
00038          CHARACTER*1 CODE
** ORDINAL 3
00039          CHARACTER*8 DATE
** ORDINAL 4
00040          REAL GRADE
00041
** ORDINAL 5
00042          RELATION REL3
PRIMARY KEY 00026      PROFID FOR AREA PFILE
ALTERNATE KEY 00028    FIELD FOR AREA PFILE
PRIMARY KEY 00031      CRSID FOR AREA CRSFILE
ALTERNATE KEY 00033    PROF FOR AREA CRSFILE
PRIMARY KEY 00036      IDENT FOR AREA CFILE
ALTERNATE KEY 00037    COURSE FOR AREA CFILE
ALTERNATE KEY 00040    GRADE FOR AREA CFILE
*****
*****      RECORD MAPPING IS NOT NEEDED FOR REALM - PFILE
*****      RECORD MAPPING IS NEEDED FOR REALM - CRSFILE
*****      RECORD MAPPING IS NEEDED FOR REALM - CFILE
00043          RESTRICT CRECORD (CODE .EQ. 'C')
00044          END
00045
*****      END OF SUB-SCHEMA SOURCE INPUT

*****
RELATION 001          RELATION  STATISTICS          *****
REL3 JOINS            AREA - PFILE
                     AREA - CRSFILE
                     AREA - CFILE

-----      BEGIN SUB-SCHEMA FILE MAINTENANCE      -----

SUBSCHEMA              CHECKSUM
COMPARE                71111404530456653576

-----      END OF FILE MAINTENANCE      -----
DDLJ COMPLETE.        0 DIAGNOSTICS.
50600B CM USED.      0.146 CP SECS.

```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 5 of 8)

```

RELATION                                * SOURCE LISTING *   (80351) DDLF 1.2+538.

00001                                SUBSCHEMA RELATION,SCHEMA=UNIVERSITY
00002
00003                                ALIAS(REALM)  SFILE=STUDENT
00004                                ALIAS(RECORD) SRECORD=STUDENT-REC
00005                                ALIAS(ITEM)   STID=STUDENT-ID.STUDENT-REC
00006
00007                                ALIAS(REALM)  CFILE=CURRICULUM
00008                                ALIAS(RECORD) CRECORD=CURR-REC
00009                                ALIAS(ITEM)   CSTID=STUDENT-ID.CURR-REC
00010                                ALIAS(ITEM)   COURS=COURSE-ID.CURR-REC
00011                                ALIAS(ITEM)   PROF=PROF-NAME
00012                                ALIAS(ITEM)   CODE=COMPLETE-CODE
00013                                ALIAS(ITEM)   DATE=COMPLETE-DATE
00014
00015                                REALM SFILE
00016                                REALM CFILE
00017
00018                                RECORD SRECORD
** WITHIN SFILE
00019                                CHARACTER*11 STID
** ORDINAL    1
00020                                CHARACTER*20 MAJOR
00021
** ORDINAL    2
00022                                RECORD CRECORD
** WITHIN CFILE
00023                                CHARACTER*14 IDENT
** ORDINAL    1
00024                                CHARACTER*11 CSTID
** ORDINAL    2
00025                                CHARACTER*6  COURS
** ORDINAL    3
00026                                REAL GRADE
** ORDINAL    4
00027                                CHARACTER CODE
** ORDINAL    5
00028                                CHARACTER*8 DATE
** ORDINAL    6
00029                                INTEGER UNITS
** ORDINAL    7
00030                                RELATION REL1
PRIMARY KEY 00019                    STID FOR AREA SFILE
ALTERNATE KEY 00020                  MAJOR FOR AREA SFILE
PRIMARY KEY 00023                    IDENT FOR AREA CFILE
ALTERNATE KEY 00024                  CSTID FOR AREA CFILE
ALTERNATE KEY 00025                  COURS FOR AREA CFILE
ALTERNATE KEY 00026                  GRADE FOR AREA CFILE
*****
*****
00031                                RECORD MAPPING IS NEEDED FOR REALM - SFILE
00032                                RECORD MAPPING IS NEEDED FOR REALM - CFILE
00033                                RESTRICT CRECORD(CODE.EQ.'C')
*****
*****                                END
*****                                END OF SUB-SCHEMA SOURCE INPUT

*****                                RELATION  STATISTICS                                *****
RELATION 001                        REL1 JOINS                                AREA - SFILE
                                           AREA - CFILE

-----                                BEGIN SUB-SCHEMA FILE MAINTENANCE                                -----

SUBSCHEMA                                CHECKSUM
RELATION                                76710464332261536703

-----                                END OF FILE MAINTENANCE                                -----
DDLF COMPLETE.                        0 DIAGNOSTICS.
50500B CM USED.                        0.128 CP SECS.

```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 6 of 8)

ADMISSIONS

* SOURCE LISTING * (80351) DDLF 1.2+538.

```

00001      SUBSCHEMA ADMISSIONS,SCHEMA=UNIVERSITY
00002
00003      ALIAS(RECORD) CRSREC=COURSE-REC
00004      ALIAS(ITEM)   CID=COURSE-ID.COURSE-REC
00005      ALIAS(ITEM)   NAME=COURSE-NAME
00006      ALIAS(ITEM)   PREREQ=PREREQUISITE
00007
00008      ALIAS(REALM)   CFILE=CURRICULUM
00009      ALIAS(RECORD) CURREC=CURR-REC
00010      ALIAS(ITEM)   STUDENT=STUDENT-ID
00011      ALIAS(ITEM)   CCID=COURSE-ID.CURR-REC
00012      ALIAS(ITEM)   CODE=COMPLETE-CODE
00013
00014      REALM COURSE
00015      REALM CFILE
00016
00017      RECORD CRSREC
** WITHIN COURSE
00018      CHARACTER*6 CID
** ORDINAL 1
00019      CHARACTER*20 NAME
** ORDINAL 2
00020      CHARACTER*20 SCHOOL
** ORDINAL 3
00021      CHARACTER*6 PREREQ
** ORDINAL 4
00022      INTEGER UNITS
00023
** ORDINAL 5
00024      RECORD CURREC
** WITHIN CFILE
00025      CHARACTER*14 IDENT
** ORDINAL 1
00026      CHARACTER*11 STUDENT
** ORDINAL 2
00027      CHARACTER*6 CCID
** ORDINAL 3
00028      CHARACTER CODE
** ORDINAL 4
00029      END
00030
*****
END OF SUB-SCHEMA SOURCE INPUT

PRIMARY KEY 00018      CID FOR AREA COURSE
PRIMARY KEY 00025      IDENT FOR AREA CFILE
ALTERNATE KEY 00026     STUDENT FOR AREA CFILE
ALTERNATE KEY 00027     CCID FOR AREA CFILE
*****
RECORD MAPPING IS NEEDED FOR REALM - COURSE
*****
RECORD MAPPING IS NEEDED FOR REALM - CFILE

```

```

-----
SUBSCHEMA                                CHECKSUM
ADMISSIONS                               56065313377542307610
-----

-----
END OF FILE MAINTENANCE
O DIAGNOSTICS.
500008 CM USED.    0.109 CP SECS.
-----

```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 7 of 8)

BURSAR

* SOURCE LISTING * (80351) DDLF 1.2+538.

```

00001      SUBSCHEMA BURSAR,SCHEMA=UNIVERSITY
00002
00003      ALIAS(RECORD) STREC=STUDENT-REC
00004      ALIAS(ITEM)   STID=STUDENT-ID.STUDENT-REC
00005      ALIAS(ITEM)   NAME=STUDENT-NAME
00006      ALIAS(ITEM)   ADDR=STREET-ADDRESS
00007      ALIAS(ITEM)   ZIP=ZIP-CODE
00008
00009      ALIAS(REALM)   ACCOUNT=ACCOUNTING
00010      ALIAS(RECORD) ACCTREC=ACCT-REC
00011      ALIAS(ITEM)   ASTID=STUDENT-ID.ACCT-REC
00012      ALIAS(ITEM)   LAB=LAB-FEES
00013      ALIAS(ITEM)   MISC=MISC-FEES
00014
00015      REALM STUDENT
00016      REALM ACCOUNT
00017
00018      RECORD STREC
** WITHIN STUDENT
00019      CHARACTER*11 STID
** ORDINAL 1
00020      CHARACTER*30 NAME
** ORDINAL 2
00021      CHARACTER*20 ADDR
** ORDINAL 3
00022      CHARACTER*10 CITY
** ORDINAL 4
00023      CHARACTER*2 STATE
** ORDINAL 5
00024      CHARACTER*5 ZIP
00025
** ORDINAL 6
00026      RECORD ACCTREC
** WITHIN ACCOUNT
00027      CHARACTER*11 ASTID
** ORDINAL 1
00028      REAL TUITION(16)
** ORDINAL 2
00029      REAL LAB(16)
** ORDINAL 3
00030      REAL BOOKS(16)
** ORDINAL 4
00031      REAL MISC(16)
00032
** ORDINAL 5
00033      RELATION REL2
PRIMARY KEY 00019      STID FOR AREA STUDENT
PRIMARY KEY 00027      ASTID FOR AREA ACCOUNT
*****
*****      RECORD MAPPING IS NEEDED FOR REALM - STUDENT
*****      RECORD MAPPING IS NOT NEEDED FOR REALM - ACCOUNT
00034      END
*****      END OF SUB-SCHEMA SOURCE INPUT

*****
RELATION 001      RELATION  STATISTICS
REL2 JOINS        AREA - STUDENT
                  AREA - ACCOUNT

-----
                  BEGIN SUB-SCHEMA FILE MAINTENANCE
-----

SUBSCHEMA
BURSAR
CHECKSUM
16643753141007716046

-----
END OF FILE MAINTENANCE
0 DIAGNOSTICS.
0.107 CP SECS.
DDLF COMPLETE.
50500B CM USED.

```

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 8 of 8)


```

Job statement
USER statement
CHARGE statement
DEFINE(QULIB/CT=PU,M=W)
ATTACH(UNIVERS)
DDL3(QC,SB=QULIB,SC=UNIVERS)
End-of-record

TITLE DIVISION.
SS  CREATES WITHIN UNIVERSITY.

REALM DIVISION.
RD  ALL.

RECORD DIVISION.

01  PROF-REC.
    03  PROF-ID          PIC X(8).
    03  PROF-NAME        PIC X(30).
    03  ACADEMIC-FIELD   PIC X(20).

01  COURSE-REC.
    03  COURSE-ID        PIC X(6).
    03  COURSE-NAME      PIC X(20).
    03  SCHOOL           PIC X(20).
    03  PROF-ID          PIC X(8).
    03  PREREQUISITE     PIC X(6).
    03  UNITS            PIC 9(2).

01  STUDENT-REC.
    03  STUDENT-ID       PIC X(11).
    03  STUDENT-NAME     PIC X(30).
    03  STREET-ADDRESS   PIC X(20).
    03  CITY             PIC X(10).
    03  STATE            PIC A(2).
    03  ZIP-CODE         PIC X(5).
    03  PHONE            PIC X(12).
    03  MAJOR            PIC X(20).

01  CURR-REC.
    03  IDENT            PIC X(14).
    03  STUDENT-ID       PIC X(11).
    03  COURSE-ID        PIC X(6).
    03  GRADE            PIC 9V9.
    03  COMPLETE-CODE    PIC A.
    03  COMPLETE-DATE    PIC X(8).
    03  UNITS            PIC 9V9.

01  ACCT-REC.
    03  STUDENT-ID       PIC X(11).
    03  TUITION          PIC 9(5)V99  USAGE IS COMP-2
                                         OCCURS 16 TIMES.
    03  LAB-FEES         PIC 9(4)V99  USAGE IS COMP-2
                                         OCCURS 16 TIMES.
    03  BOOKS            PIC 9(4)V99  USAGE IS COMP-2
                                         OCCURS 16 TIMES.
    03  MISC-FEES        PIC 9(4)V99  USAGE IS COMP-2
                                         OCCURS 16 TIMES.

End-of-record

End-of-information

```

Figure C-3. The Query Update Sub-Schema Library

```

Job statement
USER statement
CHARGE statement
DEFINE(MSTRDIR/CT=PU)
ATTACH(UNIVERS)
ATTACH(SSLIB)
ATTACH(QULIB)
DBMSTRD(NMD=MSTRDIR,LD)
End-of-record

SCHEMA NAME IS UNIVERSITY
FILE NAME IS UNIVERS
JOURNAL LOGGING ON LOGFILE.

AREA NAME IS PROFESSOR
PFN IS "PROFESS" UN IS "DBID001"
INDEX FILE ASSIGNED
PFN "PNDX" UN IS "DBID001".

AREA NAME IS COURSE
PFN IS "COURSE" UN IS "DBID001"
INDEX FILE ASSIGNED
PFN "CRSNDX" UN IS "DBID001".

AREA NAME IS STUDENT
PFN IS "STUDENT" UN IS "DBID001"
INDEX FILE ASSIGNED
PFN "SNDX" UN IS "DBID001".

AREA NAME IS CURRICULUM
PFN IS "CURRICU" UN IS "DBID001"
LOG BEFORE IMAGE RECORDS
INDEX FILE ASSIGNED
PFN "CRNDX" UN IS "DBID001".

AREA NAME IS ACCOUNTING
PFN IS "ACCOUNT" UN IS "DBID001".

SUBSCHEMA NAME IS AVERAGE
FILE NAME IS SSLIB.

SUBSCHEMA NAME IS COMPARE
FILE NAME IS SSLIB.

SUBSCHEMA NAME IS RELATION
FILE NAME IS SSLIB.

SUBSCHEMA NAME IS ADMISSIONS
FILE NAME IS SSLIB.

SUBSCHEMA NAME IS BURSAR
FILE NAME IS SSLIB.

SUBSCHEMA NAME IS CREATES
FILE NAME IS QULIB.
End-of-record

End-of-information

```

Figure C-4. The Master Directory Build

```

Job statement
USER statement
CHARGE statement
DEFINE(PROFESS,PNDX/CT=PU,M=W)
DEFINE(COURSE,CRSNDX/CT=PU,M=W)
DEFINE(STUDENT,SNDX/CT=PU,M=W)
DEFINE(CURRICU,CRNDX/CT=PU,M=W)
DEFINE(ACCOUNT/CT=PU,M=W)
RETURN(PROFESS,PNDX,COURSE,CRSNDX)
RETURN(STUDENT,SNDX,CURRICU,CRNDX,ACCOUNT)
QU(I=INPUT)
End-of-record

CREATE PROFESSOR OF CREATES FROM LIBRARY QULIB(UN=DBID001)
STORE PROF-REC SETTING PROF-ID PROF-NAME ACADEMIC-FIELD

```

```

$CRLN0080$ $CARLIN, W.L.$ $HISTORY$
$DVS00575$ $DAVIS, M.E.$ $PSYCHOLOGY$
$JCKSN750$ $JACKSON, U.B.$ $BUSINESS$
$JMS00160$ $JAMES, H.L.$ $PSYCHOLOGY$
$MLN00840$ $MALONE, R.E.$ $HISTORY$
$RSS00860$ $ROSS, W.R.$ $BUSINESS$
$SMTH0455$ $SMITH, P.R.$ $MATHEMATICS$
$WLSN0855$ $WILSON, G.R.$ $CHEMISTRY$
$YMD00170$ $YMADA, J.V.$ $BUSINESS$
*END

```

```

CREATE COURSE OF CREATES FROM LIBRARY QULIB(UN=DBID001)
STORE COURSE-REC SETTING COURSE-ID COURSE-NAME SCHOOL +
PROF-ID PREREQUISITE UNITS

```

```

$CHM103$ $BIOCHEMISTRY$ $SCIENCES$ $CRLN0080$ $CHM005$ 3
$CHM005$ $QUANTITATIVE ANAL$ $SCIENCES$ $WLSN0855$ $N/A$ 4
$CHM110$ $LINEAR OPTIMIZATIONS$ $SCIENCES$ $WLSN0855$ $MATH10$ 3
$PSY136$ $SOCIAL PSYCHOLOGY$ $LIBERAL ARTS$ $JMS00160$ $N/A$ 3
$PSY002$ $GENERAL PSYCHOLOGY$ $LIBERAL ARTS$ $JMS00160$ $N/A$ 3
$PSY003$ $PSYCHONOMICS$ $LIBERAL ARTS$ $DVS00575$ $PSY002$ 3
$HIS103$ $GREEK HISTORY$ $LIBERAL ARTS$ $MLN00840$ $N/A$ 3
$BUS017$ $CONSUMER LAWS$ $BUSINESS ADMIN$ $JCKSN750$ $N/A$ 3
$BUS001$ $ACCOUNTING I$ $BUSINESS ADMIN$ $YMD00170$ $N/A$ 3
$BUS002$ $ACCOUNTING II$ $BUSINESS AMIN$ $RSS00860$ $BUS001$ 3
$MATH10$ $COLLEGE ALGEBRA$ $SCIENCES$ $SMTH0455$ $N/A$ 3
*END

```

```

CREATE STUDENT OF CREATES FROM LIBRARY QULIB(UN=DBID001)
STORE STUDENT-REC SETTING STUDENT-ID STUDENT-NAME +
STREET-ADDRESS CITY STATE ZIP-CODE PHONE MAJOR

```

```

$122-13-6704$ $WALTER HILL$ $1960 MONTANA ST.$ +
$MINNEAPOLIS$ $MN$ $55112$ $612-143-1760$ $HISTORY$
$100-22-5860$ $GUY RICHARDS$ $143 E. LAKE BLVD.$ +
$MINNEAPOLIS$ $MN$ $55440$ $612-715-9187$ $BIOLOGY$
$124-33-5780$ $BARBARA YOUNG$ $413 MAPLE AVE.$ +
$MINNEAPOLIS$ $MN$ $55440$ $612-731-4632$ $HISTORY$
$120-44-3760$ $JERI ADAMS$ $1400 W. OAK LANES$ +
$MINNEAPOLIS$ $MN$ $55112$ $612-625-7913$ $CHEMISTRY$
$553-89-2021$ $PAUL JOHNSON$ $137 MARKET ST.$ +
$ST. PAUL$ $MN$ $55104$ $612-649-1377$ $PSYCHOLOGY$
$687-14-2100$ $PATRICIA ANDREWS$ $100 KAUREL DR.$ +
$MINNEAPOLIS$ $MN$ $55112$ $612-436-8750$ $BIOLOGY$
$197-11-2140$ $CAREN NIELSON$ $12 MORRIS ST.$ +
$ST. PAUL$ $MN$ $55104$ $612-136-9800$ $CHEMISTRY$
$678-12-1144$ $MARK PETERSEN$ $1372 PARKVIEW DR.$ +
$MINNIAPOLIS$ $MN$ $55112$ $612-143-9877$ $PSYCHOLOGY$
$387-14-1232$ $LLOYD DAVIS$ $692 FIRST ST. APT. 1$ +
$ST. PAUL$ $MN$ $55104$ $612-993-4773$ $CHEMISTRY$
$437-56-8943$ $JANET ANDERSON$ $986 SINCLAIRE AVE.$ +
$MINNEAPOLIS$ $MN$ $55112$ $612-997-6160$ $BIOLOGY$
*END

```

Figure C-5. The Query Update Data Base Creation Program (Sheet 1 of 2)

CREATE CURRICULUM OF CREATES FROM LIBRARY QULIB(UN=DBID001)
 ACCESS KEY IS \$XX99\$ ON OUTPUT FOR AREA CURRICULUM
 STORE CURR-REC SETTING IDENT STUDENT-ID COURSE-ID +
 GRADE COMPLETE-CODE COMPLETE-DATE UNITS

\$122-13-6704-01\$	\$122-13-6704\$	\$HIS103\$	4.0	\$C\$	\$09/22/80\$	3.0
\$122-13-6704-02\$	\$122-13-6704\$	\$PSY136\$	0.0	\$I\$	\$ \$ \$ 3.0	
\$100-22-5860-01\$	\$100-22-5860\$	\$SCHM103\$	3.0	\$C\$	\$05/30/79\$	3.0
\$100-22-5860-02\$	\$100-22-5860\$	\$SCHM005\$	4.0	\$C\$	\$09/22/79\$	4.0
\$100-22-5860-03\$	\$100-22-5860\$	\$MATH10\$	3.5	\$C\$	\$05/18/79\$	3.0
\$100-22-5860-04\$	\$100-22-5860\$	\$PSY002\$	0.0	\$I\$	\$ \$ \$ 3.0	
\$124-33-5780-01\$	\$124-33-5780\$	\$HIS103\$	3.5	\$C\$	\$09/22/80\$	3.0
\$124-33-5780-02\$	\$124-33-5780\$	\$BUS001\$	4.0	\$C\$	\$02/24/80\$	3.0
\$124-33-5780-03\$	\$124-33-5780\$	\$BUS002\$	4.0	\$C\$	\$09/22/80\$	3.0
\$120-44-3760-01\$	\$120-44-3760\$	\$SCHM005\$	2.0	\$C\$	\$09/22/79\$	4.0
\$120-44-3760-02\$	\$120-44-3760\$	\$SCHM103\$	3.0	\$C\$	\$05/30/80\$	3.0
\$120-44-3760-03\$	\$120-44-3760\$	\$MATH10\$	4.0	\$C\$	\$05/30/80\$	3.0
\$120-44-3760-04\$	\$120-44-3760\$	\$SCHM110\$	3.5	\$C\$	\$09/22/80\$	3.0
\$553-89-2021-01\$	\$120-44-3760\$	\$PSY136\$	3.5	\$C\$	\$05/30/80\$	3.0
\$553-89-2021-02\$	\$553-89-2021\$	\$PSY002\$	4.0	\$C\$	\$05/30/80\$	3.0
\$553-89-2021-03\$	\$553-89-2021\$	\$PSY003\$	3.5	\$C\$	\$09/22/80\$	3.0
\$687-14-2100-01\$	\$687-14-2100\$	\$SCHM005\$	4.0	\$C\$	\$09/22/79\$	4.0
\$687-14-2100-02\$	\$687-14-2100\$	\$SCHM103\$	3.5	\$C\$	\$05/30/80\$	3.0
\$687-14-2100-03\$	\$687-14-2100\$	\$MATH10\$	4.0	\$C\$	\$05/30/80\$	3.0
\$687-14-2100-04\$	\$687-14-2100\$	\$SCHM110\$	4.0	\$C\$	\$05/30/80\$	3.0
\$197-11-2140-01\$	\$197-11-2140\$	\$SCHM005\$	3.0	\$C\$	\$05/30/80\$	4.0
\$197-11-2140-02\$	\$197-11-2140\$	\$SCHM103\$	3.5	\$C\$	\$09/22/80\$	3.0
\$197-11-2140-03\$	\$197-11-2140\$	\$SCHM110\$	4.0	\$C\$	\$09/22/80\$	3.0
\$197-11-2140-04\$	\$197-11-2140\$	\$MATH10\$	4.0	\$C\$	\$05/30/80\$	3.0
\$678-12-1144-01\$	\$678-12-1144\$	\$PSY136\$	4.0	\$C\$	\$05/30/80\$	3.0
\$678-12-1144-02\$	\$678-12-1144\$	\$BUS017\$	4.0	\$C\$	\$05/30/80\$	3.0
\$678-12-1144-03\$	\$678-12-1144\$	\$HIS103\$	0.0	\$I\$	\$ \$ \$ 3.0	
\$387-14-1232-01\$	\$387-14-1232\$	\$MATH10\$	0.0	\$I\$	\$ \$ \$ 3.0	
\$387-14-1232-02\$	\$387-14-1232\$	\$SCHM005\$	0.0	\$I\$	\$ \$ \$ 4.0	
\$387-14-1232-03\$	\$387-14-1232\$	\$PSY136\$	0.0	\$I\$	\$ \$ \$ 3.0	
\$387-14-1232-04\$	\$387-14-1232\$	\$BUS017\$	0.0	\$I\$	\$ \$ \$ 3.0	
\$437-56-8943-01\$	\$437-56-8943\$	\$MATH10\$	3.0	\$C\$	\$05/30/80\$	3.0
\$437-56-8943-02\$	\$437-56-8943\$	\$SCHM005\$	3.5	\$C\$	\$05/30/80\$	4.0
\$437-56-8943-03\$	\$437-56-8943\$	\$PSY002\$	3.5	\$C\$	\$05/30/80\$	3.0

*END

CREATE ACCOUNTING OF CREATES FROM LIBRARY QULIB(UN=DBID001)
 STORE ACCT-REC SETTING STUDENT-ID

\$122-13-6704\$
 \$100-22-5860\$
 \$124-33-5780\$
 \$120-44-3760\$
 \$553-89-2021\$
 \$687-14-2100\$
 \$197-11-2140\$
 \$678-12-1144\$
 \$387-14-1232\$
 \$437-56-8943\$

*END

END

End-of-record

End-of-information

Figure C-5. The Query Update Data Base Creation Program (Sheet 2 of 2)

INDEX

- Access control B-1
- Actual key file organization 1-3, 3-12, B-1
- Advanced Access Methods (AAM) B-1
- Alias 2-1, 3-10, B-1
- Alternate key
 - Listed in sub-schema 2-3, 3-10
 - Multiple-index processing 1-3
 - Order sequenced 3-14
 - READ statement 3-6
 - START statement 3-7, 3-14
- Area (see also Data base file) B-1
- Basic Access Methods (BAM) B-1
- CDCS
 - Definition B-1
 - Description 1-2
 - Interface
 - Establish 3-2
 - Terminate 3-4
 - Locking mechanisms 4-6
- CDCS Batch Test Facility 1-2, B-1
- CDCSBTF control statement 6-1
- Child record occurrence
 - Constraint 4-4
 - Definition B-1
 - Relation 3-10, 3-13, 3-15
- CLOSE statement 3-4
- Common item 3-8, 3-10, B-1
- Concurrency 1-4, B-1
- Constraint
 - Avoiding violations 4-4
 - Definition B-1
 - Description 1-4, 4-4
- Control break
 - Definition B-1
 - Description 3-13
 - Information returned in status block 4-2, 5-12
- Control statements
 - CDCSBTF 6-1
 - DML 5-1
 - FTNS 5-1, 6-1
 - LDSET 5-1
- CYBER Database Control System (see CDCS)
- CYBER Record Manager (CRM)
 - Definition B-1
 - Description 1-3
 - Interface 1-3
- Data administrator
 - Definition B-1
 - Description 1-1
 - Responsibilities 3-1, 5-1, C-1
- Data base B-1
- Data base file (see also Realm)
 - Accessing 3-1
 - Attached by CDCS 3-2
 - Creating 3-5, 4-5
 - Direct access B-2
 - Error checking 4-2
 - File B-2
- Data base file (Contd)
 - Lock
 - Description 4-6
 - LOCK statement 3-3
 - Use 3-7, 3-8, 4-6
 - Manipulating 3-5
 - Organization 1-3
 - Position 3-6, 4-3, 5-7
 - Privacy 1-4
 - Processing considerations
 - Constraint 4-4
 - Deadlock 4-6
 - Relation 3-10, 3-15
 - Processing function 3-1, 4-3
 - Status checking 4-2
- Data base procedures 1-3, B-1
- Data base status block
 - Content 4-3
 - Definition B-1
 - Establishing in FORTRAN program 4-2
 - Test for constraint violation 4-6
 - Test for control break 4-2, 5-11
 - Test for deadlock 4-7
 - Test for end-of-file 5-7
 - Test for null occurrence 4-2, 5-11
- Data Description Language (see DDL)
- Data item B-1
- Data Manipulation Language (see DML)
- DDL 1-1, B-1
- Deadlock 4-4, B-1
- DELETE statement 3-8
- Direct access file organization 1-3, 3-12, B-2
- DML
 - Control statement 5-1
 - Definition B-1
 - Description 2-1
 - Language components 2-1
 - Preprocessor 5-1
 - Statement positioning 2-1
 - Statements 2-4, 3-1
 - Syntax requirements 2-1
- DML statements
 - CLOSE statement
 - Realm 3-4
 - Relation 3-10
 - DELETE statement 3-8
 - INVOKE statement 3-2
 - LOCK statement 3-4, 4-6
 - OPEN statement
 - Realm 3-3
 - Relation 3-10
 - PRIVACY statement 3-3, 3-10
 - READ statement
 - Realm 3-6
 - Relation 3-12
 - REWRITE statement 3-7
 - START statement
 - Realm 3-6
 - Relation 3-14
 - SUBSCHEMA statement 3-1
 - TERMINATE statement 3-4
 - UNLOCK statement 3-4
 - WRITE statement 3-5

DMLDBST routine 4-2
 DMLRPT routine 4-4
 DMS-170
 Description 1-1
 Feature summary 1-5

End option 4-1
 End-of-file 3-12, 4-1, 5-7
 EOF (see End-of-file)
 ERR option 4-1
 Error processing 4-1
 Examples
 Data Base C-1
 FORTRAN application programs 5-3
 Sub-schemas 2-1, 3-2, 3-11, C-1

File (see Data base file)
 FORTRAN DML (see DML)
 FORTRAN source program
 Developing 2-1, 3-1, 5-1
 Compiling and Executing 5-1
 Sample programs 5-3
 FTN5 control statement 5-1, 6-1

Hierarchical tree structure 3-9, B-2
 Home block B-2

I (mode) 3-3, 3-10
 Indexed sequential file organization 1-3, 3-12, B-2
 INVOKE statement 3-2
 IO (mode) 3-3, 3-10
 Item ordinal 2-2, 4-3

KEY option
 READ statement 3-6, 3-12
 START statement 3-6, 3-14
 Keyword B-2

LDSET control statement 5-1
 Listing control 5-3
 LOCK statement 3-14
 Log files
 Definition B-2
 Recovery point 4-4
 Used with CDCS 1-4
 Used with CDCS Batch Test Facility 6-1

Mapping 2-2, B-2
 Master directory
 Definition B-2
 Sample C-1
 Used with CDCS 1-2
 Used with CDCS Batch Test Facility 6-1

MODE option
 Creating a file 3-5
 OPEN statement 3-3, 3-10
 PRIVACY statement 3-3
 Relation processing 3-10

Multiple index
 Processor B-2
 Processing 1-3

Null record occurrence
 Definition B-1
 Description 3-13
 Information returned in status block 4-2, 5-12
 Null values 3-5

O (mode) 3-5
 OPEN statement
 Realm 3-3
 Relation 3-10

Parent record occurrence
 Constraint 4-4
 Definition B-2
 Relation 3-9, 3-13, 3-15
 Permanent file B-2
 Primary key
 DELETE statement 3-8
 Listed in sub-schema 2-2, 3-10
 Order sequenced 3-12
 READ statement 3-6, 3-13
 REWRITE statement 3-7
 START statement 3-7, 3-14
 Privacy key
 Definition B-2, 2-1
 Description 1-4
 Specification requirements 3-3
 PRIVACY statement 3-3, 3-10

Query Update program C-1

Rank
 Control break status 3-13
 Definition B-2
 Description 3-9
 Null record status 3-13
 Returned in status block 4-3

READ
 Locking mechanism 4-6
 Random 3-6, 3-13, 5-5
 READ statement
 Realm 3-6
 Relation 3-12
 Sequential 3-6, 3-12, 5-4
 Realm (see also Data base file)
 Definition B-2
 Listed in sub-schema 2-2, 3-1, 3-10
 Name returned in status block 4-3
 Realm lock
 Description 4-6
 LOCK statement 3-3
 Use when updating 3-7, 3-8
 Realm ordinal B-2
 Record
 Definition B-2
 Ordered stored 1-3
 Record lock
 Description 4-6
 Use when updating 3-7, 3-8
 Record occurrence 3-9, 3-12, B-2
 Record type B-2
 Recovery 1-4, B-2
 Recovery point 4-4, B-2
 Relation
 Accessing 3-8
 Definition B-2
 Description 1-4
 Listed in sub-schema 3-10
 Processing considerations 3-10, 3-15
 Processing function 3-9
 Structure 3-9
 Relation occurrence 3-8, 3-12, B-2
 Restriction
 Definition B-3
 Description 1-4
 Listed in sub-schema 2-3, 3-10
 REWRITE statement 3-7
 Root realm 3-9, B-3

Schema 1-1, B-3, C-1

START statement

Realm 3-6

Relation 3-14

Status block (see Data base status block)

SUBSCHEMA statement 3-1

Sub-schema

Definition B-3

Description 1-1

Item ordinal 2-2, 4-3

Samples 2-1

Used in application programs 3-1, 3-10, C-1

Sub-schema item ordinal 2-2, B-3

Sub-schema library 2-1, B-3

Subscripting 5-15

Subroutine 5-9

TERMINATE statement 3-4

UNLOCK statement 3-4

Updating consideration 3-15

WRITE statement 3-5

O

O

O

O

O

O

O

COMMENT SHEET

MANUAL TITLE: FORTRAN Data Base Facility Version 1
Application Programming User's Guide

PUBLICATION NO.: 60483500

REVISION: A

NAME:

COMPANY:

STREET ADDRESS:

CITY:

STATE:

ZIP CODE:

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

☐ Please reply

☐ No reply necessary

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

FOLD

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS

PERMIT NO. 8241

MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division

215 Moffett Park Drive
Sunnyvale, California 94086



CUT ALONG LINE

FOLD

FOLD